MASTER EN SCIENCES INFORMATIQUES

Logiciel open source VS closed source : mesure de la sécurite

Hostaux, David

Award date:
2016

Awarding institution:
Universite de Namur

UNIVERSITÉ DE NAMUR
Faculté d'informatique
Année académique 2015-2016

Logiciel open source VS closed source : mesure de la sécurité

David Hostaux

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)

Jean-Noël Colin

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

1. L'état de l'art

Le débat entre l'open source et le closed source est un grand débat qui existe depuis longtemps et dont on peut trouver différents avis sur le web. Ces avis sont tout aussi bien « pour » l'open source et « contre » le closed source qu'inversement.

Cependant, il existe très peu de documents de littérature qui abordent ce sujet. Nous n'avons d'ailleurs pas trouvé de modèle de sécurité qui traite du sujet de l'open source VS le closed source. [Schryen & Kadura, 2009] expliquent cela par le fait que les logiciels de ces deux catégories sont multiples et touchent plusieurs domaines de l'informatique[1]. Il est donc presque impossible de passer en revue toutes ces branches différentes et de définir un modèle unique de sécurité qui permet de définir à chaque fois si l'open source ou le closed source est à recommander.

De par ce constat, pour pouvoir aborder le sujet, les auteurs des documents de littérature décident de choisir un ou plusieurs domaines de l'informatique, sélectionnent un même nombre de logiciels open source et closed source disponibles dans ce domaine et les comparent suivant un modèle de sécurité défini.

Nous partirons vers ce même type de comparaison dans ce mémoire.

Pour réaliser les comparaisons entre les logiciels, les auteurs se basent sur des modèles de sécurité ainsi que sur des bases de données de vulnérabilité. Le tout, et en fonction du modèle utilisé, permettant de mesurer le nombre de failles, le nombre de patchs effectués et des nouveaux bugs amenés par ces patchs, du temps de correction des failles… C'est donc une investigation principalement basée sur l'analyse du système opérationnel qui montre seulement un des aspects de la quantification de la sécurité logicielle. Cela pourrait donc être complété par d'autres évaluations, d'autres points de vue.

C'est pourquoi nous aborderons une partie qui se focalisera sur les hackers, leurs façons d'opérer, leurs motivations… Cela apportera un nouveau point de vue en déterminant si les attaques sont plus souvent portées sur des logiciels en source ouverte ou en source fermée.

1 Exemple : réseaux, internet, serveurs, logiciels...

2. Résumé

Le but de ce document a été, à la base, de savoir s'il était possible de déterminer lesquels, des logiciels en source fermée ou des logiciels en source ouverte, sont les plus sécurisés. Nous avons donc élaboré ce document dans l'optique de le déterminer.

Cependant, au fur et à mesure des recherches, nous avons pu nous rendre compte que définir la sécurité d'un logiciel était quelque chose de complexe et qu'il y avait beaucoup de données à prendre en compte pour pouvoir la définir.

D'où le fait que nous avons réalisé un guide à la décision de logiciel, qui permet de comparer deux logiciels du point de vue de la sécurité, en fonction d'arguments et de mesures définies. Ce guide n'est pas figé et peut évoluer en fonction de nouveaux critères de sélection. Il nous a aussi par ailleurs permis de répondre au sujet de ce document mentionné ci-dessus.

2.1. Mots clés

Open source, Closed source, mesure, analyse, sécurité logiciel, vulnérabilités, exploits, hackers

3. Avant-propos

Je tiens à remercier mon promoteur, monsieur Jean-Noël Colin, pour plusieurs raisons :

– Tout d'abord, pour avoir accepté de me suivre dans ce sujet que j'avais envie d’approfondir.

– Ensuite, pour avoir directement compris et pris en compte ma donnée géographique. En effet, habitant et travaillant à plus d'une heure de l'Université de Namur, monsieur Colin a directement adapté son suivi pour pouvoir le réaliser à distance. Cela m'a permis d'éviter de perdre du temps dans des trajets inutiles.

– Finalement, pour tous les conseils avisés qu'il m'a prodigués, aussi bien pour mes recherches de données que pour l'analyse de celles-ci. Le temps qu'il a consacré à me répondre, sa rapidité à m'envoyer les documents dont j'avais besoin et son aide apportée tout au long de la réalisation de ce document m'ont aussi été précieux.

Je tiens aussi à remercier mes parents :

– Marie-Flore Dromelet, pour les nombreuses lectures, corrections orthographiques et grammaticales qu'elle a faites tout au long de la rédaction du mémoire.

– Bernard Hostaux, pour sa relecture finale et ses conseils en tournure de phrases.

Table des matières

1. L'état de l'art....3
2. Résumé....4
2.1. Mots clés....4
3. Avant-propos....5
4. Glossaire....10
5. Introduction....12
6. Recherches de points de vue....14
6.1. Listing d'arguments « pour » et « contre »....14
6.1.1. Arguments pour l'open source....14
6.1.2. Arguments contre l'open source....14
6.1.3. Arguments pour le closed source....15
6.1.4. Arguments contre le closed source....16
6.2. Comparaison à partir de définitions....16
6.3. Nuancer les points de vue....17
6.3.1. Les entreprises par rapport à l'open source et le closed source....17
6.3.2. Le taux d'utilisation d'un logiciel....17
6.3.3. Les arguments en commun....17
6.3.4. Un jeu de croyance....17
6.3.5. Aucune donnée concrète....18
6.4. Pour aller plus loin....18
7. Analyser la sécurité d'un logiciel....19
7.1. La définition des termes....19
7.2. Propriétés de sécurité tangibles / intangibles....19
7.3. Les avis subjectifs....20
7.4. Faire une analyse des pratiques de programmation....20
8. Démarche utilisée pour notre analyse....22
8.1. Recherches de données mesurables....22
8.2. Choisir les logiciels à comparer....22

8.3. Analyse objective....23
9. Les failles publiées....25
9.1. La méthodologie....25
9.1.1. Le but de l'analyse....25
9.1.2. Recueillir les données....25
9.1.3. Mettre un poids sur les vulnérabilités....25
9.1.4. Travailler par rapport à un pourcentage....26
9.2. Comparaison des failles entre logiciels de la même branche informatique....27
9.2.1. Les systèmes d'exploitation....27
9.2.2. Les suites bureautiques....28
9.2.3. Les serveurs web....29
9.3. Comparaison des failles entre logiciels open source et closed source....30
10. Les exploits de hackers....32
10.1. La méthodologie....32
10.1.1. Le but de l'analyse....32
10.1.2. Récolter les données....32
10.1.3. Le nombre d'exploits....32
10.1.4. Le pourcentage de vérification....32
10.2. Comparaison des exploits entre logiciels de la même branche informatique....32
10.2.1. Les systèmes d'exploitation....33
10.2.2. Les suites bureautiques....34
10.2.3. Les serveurs web....35
10.3. Comparaison des failles entre logiciels open source et closed source....36
11. Les failles publiées par rapport aux exploits réalisés....38
11.1. La méthodologie....38
11.1.1. Le but de l'analyse....38
11.1.2. Récolter les données....38
11.1.3. Mettre un poids sur les exploits....38
11.1.4. Comparaison du pourcentage d'exploits par rapport à la liste des failles....38
11.2. Comparaison des exploits entre logiciels de la même branche informatique....39

11.2.1. Les systèmes d'exploitation....39
11.2.2. Les suites bureautiques....41
11.2.3. Les serveurs web....42
11.3. Comparaison des failles entre logiciels open source et closed source....43
12. Analyse des résultats....46
12.1. Les résultats....46
12.2. Analyse....46
12.2.1. Lien entre le poids des failles et le poids des exploits....46
12.2.2. Les motivations....47
12.3. D'autres documents amenant au même résultat....47
13. Les limites des résultats....49
13.1. Les logiciels open source et closed source....49
13.2. La sécurité d'un logiciel est un ensemble de composants....49
13.3. Le processus de développement....49
13.4. La licence utilisée....50
13.5. La publication des failles....50
13.6. Les exploits réalisés....51
14. Guide à la décision de logiciels....53
14.1. Conclusion concernant l'open source et le closed source....53
14.2. Le but du guide à la décision....54
14.2.1. Le guide à la décision vis-à-vis de la comparaison de logiciels open source et closed source....55
14.3. La méthodologie utilisée....55
14.3.1. Définition des catégories....55
14.3.2. Définition des arguments de sécurité....56
14.3.3. Attribution d'une mesure aux arguments....56
14.3.4. Attribution d'un poids aux mesures....58
14.3.5. Déterminer la façon d'attribuer les données....58
14.4. Encoder les données récoltées....59
14.4.1. Fonctionnement du fichier....59

14.4.2. Encodage des données....60
14.5. Définitions des arguments de la catégorie « sécurité externe »....61
14.5.1. La phase de développement du logiciel....62
14.5.2. Le pourcentage de patchs réalisés....64
14.5.3. Les exploits réalisés sur un logiciel....67
14.6. Définitions des arguments de la catégorie « sécurité interne »....69
14.6.1. Le poids des failles d'un logiciel....69
14.6.2. Les lignes de code du logiciel....72
14.7. Réaliser la comparaison des logiciels....74
14.7.1. Fonctionnement du fichier....75
14.7.2. Comparaison des données....76
14.8. Mise en pratique : comparaison des 6 logiciels étudiés....78
14.8.1. Les systèmes d'exploitation....79
14.8.2. Les suites bureautiques....79
14.8.3. Les serveurs web....80
14.8.4. Résultats de la comparaison entre logiciels open source et closed source....80
14.9. Comparer les logiciels avec le tableau de comparaison....81
14.9.1. Limite des mesures de sécurité définies....81
14.9.2. Définir les logiciels et arguments à comparer....82
14.9.3. Pouvoir comparer les arguments que l'on souhaite comparer....83
14.9.4. Ajouter un argument et/ou une mesure de sécurité....84
14.9.5. Limite des données récoltées....85
15. Conclusion....86

4. Glossaire

– C**PE (Common Platform Enumeration) :** Assignation d'un nom aux versions d'un logiciel vulnérable à des vulnérabilités identifiées dans CVE [CPE].

– **CS (Closed-Source) :** Logiciel dont le code source est non disponible pour le public [Hoepman & Jacobs, 2013].

– **CVE (Common Vulnerabilities and Exposures) :** Dictionnaire de vulnérabilités de sécurité fourni par MITRE [CVE].

– **CVE Details :** Base de données contenant les failles découvertes sur une série de logiciels. Ces failles sont répertoriées par le dictionnaire CVE [CVE details].

– **CVSS (Common Vulnerability Scoring System) :** Attribution d'un score (de 0 à 10) aux vulnérabilités en fonction de leur sévérité. Ce score est attribué par l'équipe NVD et peut changer dans le temps en fonction de différents facteurs (moindre et plus grande sévérité de la faille par exemple) [CVSS].

– **CWE (Common Weakness Enumeration) :** Dictionnaire de vulnérabilités auxquelles ont été assignés des types [CWE].

– **Exploit time :** Le temps qu'il a fallu pour arriver à exploiter une vulnérabilité [Schryen, 2011].

– **Faille :** Une faille est un événement qui se produit lorsqu'un système ne fait pas ce qu'il devrait faire [Jonsson & Stömberg & Lindskog, 1999].

– **FSF (Free Software Foundation) :** Fondation qui maintient les licences open source [FSF].

– **Injection de code :** injecter du code externe dans un software [Schryen, 2011].

– **ISO (International Standard Organisation) :** Organisation qui établit et publie des normes internationales [ISO].

– **MTBVD (Mean Time Between Vulnerability Disclosures) :** Moyenne du nombre de jours depuis la release d'un logiciel divisé par le nombre de vulnérabilités publiées [Schryen, 2011].

– **- NVD (National Vulnerability Database) :** Base de données de vulnérabilités fournies par NIST (National Institute of Standards and Technology), une agence de département du commerce des États-Unis [NVD].

– **ORD (Original Release Date) :** représente la mesure sur le dictionnaire CVE entre la divulgation de la vulnérabilité et la considération de mise en résolution. Vaut 0 lorsque la vulnérabilité n'a pas été divulguée. Lorsqu'il n'y a pas d'informations sur ce temps, le calcul entre le temps de patch (patch time) et le temps d'exploit (exploit time) contiendra des erreurs [Schryen, 2011].

– **OS (Open Source) :** Logiciel dont le code source est disponible pour le public [Hoepman & Jacobs, 2013].

– **OSI (Open Source Initiative)[2] :** Organisation qui définit un lot de critères que les logiciels doivent suivre[3], mais qui ne définit pas les personnes qui sont éligibles pour le faire [OSI].

– **Patch :** Modification apportée à une section de code afin d'y apporter diverses corrections [Schryen, 2011].

– **Patch d'index :** Niveau d'activité de la communauté à sortir des patchs [Schryen & Kadura, 2009].

– **Patch time :** Le temps qu'il a fallu pour corriger une vulnérabilité [Schryen & Kadura, 2009].

– **Ratio bug/ligne de code :** 1/35 [Schryen, 2011].

– **US Department of Homeland Security (Département de la Sécurité intérieure des États-Unis) :** Ce département effectue un audit journalier de ± 40 logiciels OS [USDHS].

– **Violation de sécurité :** Incident dû à une vulnérabilité de sécurité [Schryen & Kadura, 2009].

– **Vulnérabilité :** On parle de vulnérabilité lorsque les bugs peuvent être utilisés par les hackers pour avoir accès au système et pouvoir faire ce qu'ils veulent [Schryen, 2011].

– **Vulnérabilité life cycle :** Processus orienté perspective entre les vulnérabilités et leurs patchs intégrant plusieurs états. Être conscient du fait que faire des patchs sur des vulnérabilités peut très bien créer de nouvelles vulnérabilités [Schryen, 2011].

2 OSI définit plusieurs licences qui sont maintenues par FSF
3 Exemple : permission de modifier/redistribuer le code

5. Introduction

Qui n'a jamais entendu dire « Les logiciels open source sont plus sécurisés parce que […] » ou encore « Les logiciels closed source sont plus sécurisés parce que [...] »[4] ? Pas nous en tout cas.

Nous avons d'ailleurs pu remarquer que, la plupart du temps, chacun était convaincu de ce qu'il disait, même si son avis était subjectif. Nous nous sommes donc demandé ce qu'il en était réellement et si vraiment, il était possible de déterminer une de ces deux manières de développer comme étant plus sécurisée que l'autre.

Ce fut le point de départ de ce mémoire.

Avant de se lancer dans l'analyse proprement dite, il faut tout d'abord savoir en quoi consiste une analyse de sécurité d'un logiciel et la difficulté que cela peut représenter. Cela permet de pouvoir déterminer la démarche que nous utiliserons tout au long de nos analyses.

Il faut aussi sélectionner les logiciels à analyser. En ce qui nous concerne, nous avons décidé de comparer 3 logiciels open source[5] et 3 logiciels closed source[6].

En ce qui concerne l'analyse, comme le montrent [Schryen & Kadura, 2009], bien souvent, la sécurité d'un logiciel est vue comme un seul atome. Cependant, il faut découper cet atome en plusieurs composants[7] qu'il faut analyser pour pouvoir déterminer la sécurité ou non d'un logiciel[8].

Comme il est presque impossible de passer en revue tous ces composants[9], nous avons décidé de réaliser notre analyse par rapport à trois éléments:

- **Les failles d'un logiciel.** Nous comparerons les logiciels open source et closed source en fonction du nombre de failles et vulnérabilités qui ont été trouvées.
- **Les attaques des hackers.** Nous allons nous intéresser aux motivations des hackers et aux attaques qu'ils portent pour voir s'il est possible de déterminer s'il est plus attrayant pour eux d'attaquer un logiciel open source ou closed source.
- **La relation qu'il peut y avoir entre les attaques des hackers et les failles d'un logiciel.** Nous allons faire une relation entre les attaques portées sur un logiciel et les vulnérabilités que celui-ci comporte pour voir, entre autres, si les vulnérabilités des logiciels en source ouverte ou fermée sont plus souvent attaquées que l'autre.

4 Vous pouvez remplacer les […] par n'importe quelle excuse subjective.
5 Le noyau Linux, Libre Office, Apache HTTP Serveur
6 Windows 7, Microsoft Office, IIS Serveur
7 Fiabilité, confidentialité, intégrité, etc
8 Exemples : Quand est-ce qu'un logiciel est fiable ? Quels sont les éléments qui peuvent le rendre non fiable ? Quelles données concrètes puis-je trouver pour prouver cette fiabilité ? …
9 Pour les raisons expliquées dans le point 1 de ce mémoire : « l'état de l'art »

Une fois les différentes données récoltées, nous en ferons leurs analyses. Nous déterminerons ainsi s'il est possible de définir un type d'ouverture de code recommandable du point de vue de la sécurité ou non.

Pour terminer, nous établirons un guide à la décision. Son but sera de pouvoir comparer deux logiciels en fonction de son niveau de sécurité, qu'il soit en source ouverte ou non, suivant des arguments définis de la manière la plus objective possible et mesurable.

En ce qui concerne notre analyse, ce guide nous permettra de confirmer, ou non, les résultats que nous avons obtenus auparavant.

6. Recherches de points de vue

Afin de lancer le débat concernant la sécurité entre les logiciels open source et closed source, nous allons, dans cette première partie, mettre en évidence différents points de vue que l'on peut trouver sur le sujet. Cela nous permettra de nous faire un premier avis sur la manière d'aborder le sujet.

6.1. Listing d'arguments « pour » et « contre »

Dans ce premier point, nous allons établir un listing[10] de différents arguments[11] que nous allons classer comme étant « pour » ou « contre » la manière de développer un logiciel en open source ou en closed source.

6.1.1. Arguments pour l'open source

– **Accès au code source veut dire plus de lecteurs.** C'est une idée assez répandue concernant l'open source : le fait que le code source d'un logiciel est ouvert, donc disponible à tout le monde, signifie qu'il peut y avoir potentiellement beaucoup de lecteurs. [Schryen & Kadura, 2009] mettent en évidence l'importance de cette communauté car plus il y a de lecteurs, plus il y a de chances de trouver et de pouvoir corriger des problèmes de sécurité.

– **Tout le monde peut soumettre des corrections.** [Schryen & Kadura, 2009] montrent aussi l'importance de la communauté par le fait que chacun peut corriger des vulnérabilités et soumettre des corrections au gestionnaire du logiciel. On n'est donc pas dépendant d'une compagnie et l'on ne doit pas attendre que celle-ci fasse la correction nécessaire. [Hoepman & Jacobs, 2013] appuient cet argument en disant qu'il est alors plus facile pour ce gestionnaire d'effectuer des patchs sur le logiciel qu'il gère.

– **Augmente l'excellence du code.** Selon [Hoepman & Jacobs, 2013], en choisissant de rendre disponible le code de leurs logiciels, les développeurs savent que leurs sources seront lues par d'autres. Cela les pousse alors à utiliser de bonnes pratiques de développements, à avoir un code propre, nettoyé, de meilleure qualité, etc. [Walden & Doyle & Welch & Whelan, 2009] montrent que ces bonnes pratiques peuvent avoir un impact sur la sécurité et que cela baisse la densité de vulnérabilités d'un logiciel.

– **Les logiciels open source sont moins utilisés et donc, moins hackés.** C'est une idée que l'on retrouve assez souvent, surtout pour les systèmes d'exploitation ayant un noyau Linux : vu qu'ils ne sont pas beaucoup utilisés[12], les hackers ont moins d'intérêt à lancer des attaques sur ces systèmes vu qu'ils toucheront beaucoup moins de monde.

6.1.2. Arguments contre l'open source

– **Beaucoup de contributeurs ne veut pas dire beaucoup de lecteurs.** Ce n'est pas parce qu'un projet a beaucoup de contributeurs que chacun a lu, analysé et testé l'entièreté du logiciel.

10 Ce listing n'a pas pour but d'être exhaustif.
11 Ces arguments sont pour la plupart subjectifs et n'ont rien de scientifique.
12 Par rapport à des systèmes d'exploitation comme Windows ou Mac OS

Ceux-ci pourraient penser que cela ne sert à rien de lire et analyser le code vu que d'autres l'ont déjà réalisé.

– **Une différence d'expertise entre les contributeurs.** [Schryen & Kadura, 2009] avancent le fait que s'il y a beaucoup de contributeurs, cela ne veut pas dire que ces derniers ont tous la même expertise, le même niveau de compétences ni le même niveau de connaissances. Ces compétences qu'il faut posséder pour assurer la sécurité d'un logiciel peuvent donc faire défaut avec l'open source étant donné qu'il n'y a, bien souvent, pas de sélection de programmeurs pour que ceux-ci puissent participer au développement. Dans ce cas, on ne peut pas être certain que ce seront des personnes qualifiées qui développeront et inspecteront le code ou même que des hackers ne se font pas passer pour des contributeurs du projet.

– **Les contributeurs doivent avoir d'autres compétences.** Toujours en ce qui concerne les compétences, [Schryen & Kadura, 2009] montrent que pour pouvoir détecter tous les problèmes, il faut posséder d'autres compétences que le domaine du logiciel développé[13]. Comme en open source, tout le monde peut participer au développement du logiciel, il n'y a aucune garantie que les contributeurs du logiciel sont compétents dans ces domaines.

– **Corrections de toutes les failles.** Si le code source est ouvert, cela veut dire qu'il est disponible à tout le monde, même aux hackers. Ceci peut être un désavantage pour les développeurs car, comme le disent [Hoepman & Jacobs, 2013], ils vont devoir corriger toutes les failles visibles tandis qu'un hacker scrutera toujours la moindre petite faille et pourra se contenter d'en trouver une pour pouvoir attaquer.

– **Le logiciel n'est peut-être pas celui que l'on pense.** La plupart des licences open source permettent de modifier le code source et de le redistribuer. Cependant, [Schryen, 2011] nous fait remarquer que rien ne précise qui est éligible pour pouvoir le faire. Cela amène donc le problème relaté par [Hoepman & Jacobs, 2013] qui fait qu'il n'y a pas de garantie que le logiciel que l'on exécute est bien celui qu'il prétend être. On pourrait donc très bien exécuter un programme en pensant qu'il est le bon alors que c'est un logiciel piraté.

6.1.3. Arguments pour le closed source

– **La sécurité par l'obscurité.** Un des principaux arguments que l'on retrouve pour le closed source est que le fait que le code source ne soit pas disponible et donc visible par tout le monde. [Schryen & Kadura, 2009] expliquent que le code est donc « caché », que les défauts ne peuvent, normalement, pas être vus et donc, ne sont connus que lorsqu'ils sont corrigés lors des mises à jour via des patchs.

– **Les hackers peuvent être découragés par la perte de temps nécessaire pour pouvoir avoir accès à l'information.** Le fait de ne pas avoir accès au code source fait que les hackers n'ont pas accès directement à l'information et aux différentes failles. [Hoepman & Jacobs, 2013] disent que ces personnes, pas nécessairement bien intentionnées, doivent donc tout d'abord perdre leur temps pour essayer d'accéder au code et ce, sans jamais avoir la certitude d'y arriver. Quand bien

13 Des connaissances en réseau, cryptographie ou autres sont souvent nécessaires pour savoir comment contrer des attaques potentielles d'un logiciel

même ils réussiraient, il faudra alors qu'ils passent leur temps à trouver une faille. Ils ont donc plus de difficultés à lancer une attaque et pourraient être découragés, comme le disent [Floyd & Harrington & Hivale, 2007], à lancer une attaque.

– **Structure d'une entreprise.** Bien souvent, derrière les logiciels closed source se trouvent des entreprises. Le logiciel développé bénéficie donc de toute la structure de l'entreprise à laquelle il appartient[14].

6.1.4. Arguments contre le closed source

– **L'obscurité du code source ne cache pas tous les défauts.** [Schryen & Kadura, 2009] font remarquer que ce n'est pas parce que le code source d'un logiciel n'est pas disponible que ce dernier est pour autant sécurisé. En effet, il est toujours possible de simplement lancer un débogueur, un analyseur de performance, un système de tracking... pour pouvoir détecter une faille.

– **Il est toujours possible d'accéder au code source.** [Schryen & Kadura, 2009] montrent aussi que garder les sources fermées n'est pas non plus toujours facile et il est toujours possible d'arriver à désassembler un code pour accéder à ses sources. À partir de ce moment, il est possible que le hacker publie les sources et donc, que tout le monde puisse avoir accès au code, aux bugs et aux failles. Cela révélerait, à ce moment-là, à quel point le logiciel peut être sécurisé ou non.

– **Peu de lecteurs du code.** Dans ce cas-ci, on reprend l'argument de [Schryen & Kadura, 2009] sur l'importance de la communauté pour le prendre dans l'autre sens. Avec les logiciels closed source, seuls les développeurs du logiciel ont accès au code source, ce qui fait qu'il y a assez peu de lecteurs. Comme il est très difficile d'analyser soi-même ce que l'on fait, il est alors très difficile d'arriver à trouver les failles dans son propre code[15]. Les programmes qui ne sont donc pas lus par plusieurs lecteurs externes peuvent potentiellement être plus dangereux.

6.2. Comparaison à partir de définitions

[Hoepman & Jacobs, 2013] font une autre comparaison entre l'open source et le closed source : ils partent de la définition des mots suivants : la sécurité, le risque et l'exposition d'un logiciel.

– *La sécurité d'un logiciel :* mesure entre le nombre de vulnérabilités et leurs sévérités.

– *Le risque d'un logiciel :* combinaison de probabilités d'une attaque sur le logiciel et les dommages qu'ils peuvent occasionner.

– *L'exposition d'un logiciel :* probabilité d'une attaque sur le système. Cette attaque peut dépendre de plusieurs facteurs.

En prenant ces définitions pour comparer les logiciels au code ouvert et les logiciels au code fermé, on peut déduire qu'aussi bien l'un que l'autre ne modifient en rien la sécurité d'un logiciel : Cela n'introduit pas de nouveaux bugs. Par contre, pour les logiciels avec code disponible, l'exposition est augmentée par rapport à un logiciel au code non disponible : les failles sont rendues plus visibles pour les hackers. Cela augmente donc le risque d'une attaque. Plus le logiciel est

14 Exemples : le support, le suivi, la documentation...
15 Les développeurs savent ce qu'ils ont envisagé, mais ne savent pas à quoi ils n'ont pas pensé.

exposé, plus le risque est grand.

Il y a donc un désavantage pour les logiciels open source.

6.3. Nuancer les points de vue

Les différents arguments décrits ci-dessus sont assez subjectifs. C'est pourquoi il convient de ne pas croire tout ce qu'on lit et de les nuancer.

6.3.1. Les entreprises par rapport à l'open source et le closed source

Bien souvent, les personnes associent « closed source » à « entreprise » et « open source » à « une personne ou un groupe de personnes ». Ceci est bien évidemment faux et on peut très bien avoir un logiciel en closed source développé par une seule personne tout comme un logiciel open source qui est développé par une entreprise.

6.3.2. Le taux d'utilisation d'un logiciel

Le taux d'utilisation d'un logiciel ne peut pas non plus faire partie de critères fiables pour déterminer le niveau de sécurité d'un logiciel. En effet, ce n'est pas parce qu'un logiciel est peu utilisé aujourd'hui qu'il ne va pas être beaucoup plus utilisé dans le futur. Comme le montre [Furnell, 2010], c'est d'ailleurs ce qui est arrivé avec Mac : au début de leur utilisation, il basait leur communication sur le fait qu'il n'y avait aucun virus par rapport à Windows. Leur système a commencé à avoir du succès, les virus ont commencé à arriver et ils ont alors changé leur communication en disant qu'il n'y avait pas les virus PC sur les MAC[16].

Comme le montre [Seebruck, 2015], aucun logiciel n'est à l'abri de failles et personne ne peut dire que les logiciels qu'il utilise ne seront jamais hackés.

6.3.3. Les arguments en commun

Comme le font remarquer [Hoepman & Jacobs, 2013], en prenant en compte les arguments précédents, certains des « pour » ou « contre » peuvent au final être en commun à l'open source et le closed source. Par exemple: un logiciel open source peut aussi pouvoir bénéficier d'une structure d'une entreprise, un logiciel closed source peut très bien être développé par une seule personne qui n'a pas toutes les compétences nécessaires, les compétences des développeurs sont tout aussi importantes sur un logiciel open source que closed source, etc.

6.3.4. Un jeu de croyance

[Schryen & Kadura, 2009] disent que la plupart des arguments résultent d'un jeu de croyance. Ainsi, par exemple, une personne étant pour l'open source va dire que le fait que le code source soit visible est un argument positif, tout comme une personne étant pour le closed source va dire que c'est le fait qu'un code source soit caché qui est positif. Ce choix résulte simplement de la préférence de la personne en question et ne peut donc constituer un argument concret.

16 Les virus sévissant sur MAC n'étant pas « les mêmes » que ceux sévissant sur PC

6.3.5. Aucune donnée concrète

[Schryen & Kadura, 2009] montrent que pour pouvoir mesurer la sécurité d'un logiciel, il faut, entre autres, se baser sur des données concrètes. Or, aucun des arguments cités ci-dessus n'est basé sur des données concrètes. Ils ne peuvent donc pas être vérifiés et donc, être crédibles.

6.4. Pour aller plus loin

Comme vous l'aurez compris, il n'est bien évidemment pas possible de tirer une conclusion avec cette série de points de vue. Ce n'était d'ailleurs pas le but de cette partie.

Au contraire, nous allons nous servir de ces points de vue comme rampe de lancement pour la suite : pour réaliser une analyse du point de vue de la sécurité, il n'est pas possible de se baser sur des avis subjectifs.

7. Analyser la sécurité d'un logiciel

Comme l'écrivent la plupart des auteurs de littérature que nous avons pu lire, effectuer une analyse de sécurité sur un logiciel n'est pas une mince affaire. Nous allons détailler pourquoi via les quelques points ci-dessous.

7.1. La définition des termes

Tout d'abord, comme le montrent [Schryen, 2011], [Jonsson & Stömberg & Lindskog, 1999] ou encore [Colombo & Guerra & Filho & Gomes], bien souvent, en fonction de ceux qui les définissent, des termes de sécurités peuvent avoir des définitions différentes. Ce n'est donc déjà pas évident si, en utilisant le même terme, deux personnes veulent décrire des choses différentes, c'est déjà mal parti. Surtout si, comme le disent [Colombo & Guerra & Filho & Gomes], certaines définitions de termes dépendent d'autres définitions.

C'est pourquoi, dans leurs analyses, [Schryen, 2011] et [Jonsson & Stömberg & Lindskog, 1999] commencent tout d'abord par définir les termes qu'ils vont utiliser dans leurs documents. C'est donc ce que nous avons fait aussi. Vous pourrez trouver la définition des termes dans le chapitre « *4. Glossaire* » de ce mémoire[17].

7.2. Propriétés de sécurité tangibles / intangibles

Il existe différentes propriétés de sécurité[18] qui sont définies dans différents modèles de sécurité[19].

Comme nous en informent [Colombo & Guerra & Filho & Gomes], certaines de ces propriétés sont tangibles, c'est-à-dire qu'elles sont mesurables et définissables.

Par exemple :

- La disponibilité d'un logiciel. Un logiciel, disponible 23 heures sur 24 a un taux de disponibilités d'un peu moins de 96 %.

Par contre, ils nous informent qu'il existe des propriétés intangibles, c'est-à-dire, des éléments qui sont impossibles à définir avec certitude ou précision.

Par exemple :

- Comment calculer le taux de confidentialité d'un logiciel ? Quelle unité de mesure utiliser ?

Et ce sont ces éléments non mesurables qui peuvent poser problème car, comme le disent [Mellado & Fernandez-Medina & Piattini], pour pouvoir prouver quelque chose, il faut pouvoir le mesurer et le démontrer avec des nombres concrets. Selon eux, la sécurité ne peut pas être améliorée sans mesure.

17 Voir aussi l'annexe 1 « Les termes à ne pas confondre »
18 La confidentialité, l'intégrité, la fiabilité, la disponibilité, l'accessibilité, etc.
19 Modèle de qualité McCall's, modèles de qualité Boehm's, etc.

C'est pourquoi, pour les éléments intangibles, [Colombo & Guerra & Filho & Gomes] proposent d'effectuer des mesures universelles sur un logiciel et de lui récolter sa mesure. En effectuant la même opération sur un autre logiciel, il sera possible de comparer les données mesurées et ainsi, en déduire un résultat.

7.3. Les avis subjectifs

[Morrison] nous informe même qu'il existe des frameworks de mesure qui offre un outil pour collecter et comparer les données de logiciel.

Exemples pour collecter les données :

- Pour l'usage, définir le taux d'utilisation du logiciel : pas utilisé, journalier, toutes les semaines, tous les mois, etc.
- Facilité d'utilisation : est-ce que le logiciel est facile à utiliser ?

Cependant, pour répondre à certaines de ces mesures, il va falloir se servir d'avis subjectifs. En effet, pour ce deuxième exemple, il est possible qu'une personne trouve un logiciel facile à utiliser tandis qu'une autre non. Et il n'est pas possible de déterminer qui dit vrai. C'était aussi le cas, par exemple, par rapport à tous les points de vue subjectifs que nous avons cités dans le chapitre précédent.

C'est pour cela que [Schryen & Kadura, 2009] nous disent que, pour avoir une analyse correcte, il faut pouvoir faire abstraction de la personne, du rôle, de l'organisation gérant les différents logiciels choisis. Cela évite ainsi de prendre en compte des avis subjectifs ou encore d'avoir un avis ou une prise de position avant même d'avoir commencé.

Et de nouveau, pour pouvoir prouver quelque chose, il faut pouvoir le mesurer.

7.4. Faire une analyse des pratiques de programmation

La sécurité peut aussi provenir de certaines bonnes pratiques de programmation. En effet, comme le montrent [Walden & Doyle & Welch & Whelan, 2009], les techniques de programmation peuvent influencer la densité de vulnérabilités.

Par exemple :

- Éviter la complexité du code.
- La taille du code.
- Le bruit, la duplication de code.

En effet, si une faille est découverte dans un code qui a été dupliqué plusieurs fois, il faut s'assurer de faire les corrections dans chacune des duplications.

[Walden & Doyle & Welch & Whelan, 2009] nous disent que la complexité de faire une analyse par rapport aux pratiques de programmation vient du fait, comme nous l'avons déjà abordé, qu'il est assez difficile de pouvoir le mesurer avec précision. Pour cela, certains outils existent, comme

KLOC, montré par [Walden & Doyle & Welch & Whelan, 2009], ou encore ASIDE, montré par [Xie & Lipford & Chu], pour détecter le nombre de vulnérabilités suite à une analyse statique du code.

Cependant, comme le disent [Saffiotti & Awyzio & Brown, 2004], faire une analyse statique ne suffit pas. En effet, certaines vulnérabilités ne sont détectables seulement lorsque le logiciel fonctionne.

Pour pouvoir détecter les différentes vulnérabilités d'un logiciel, il faut donc pouvoir faire une analyse statique (lecture du code) et dynamique (analyse du comportement du software en état de fonctionnement) du logiciel.

8. Démarche utilisée pour notre analyse

Grâce à ces deux premiers chapitres, nous pouvons tirer des enseignements qui nous serviront lors de notre comparaison entre des logiciels open source et closed source.

8.1. Recherches de données mesurables

Avant de se lancer dans l'analyse proprement dite, il faut définir la notion de sécurité. Autrement dit, quand est-ce que l'on considère qu'un logiciel n'est pas sécurisé.

Par exemple :

- En fonction de son nombre de failles ?
- En fonction de son nombre d'intrusions ?
- En fonction de son nombre d'utilisations ?
- En fonction de sa difficulté à entrer dans le système (doit-on être logué ou non, est-ce que l'on peut faire cela à distance) ?
- Etc.

De plus, comme le disent [Schryen & Kadura, 2009] et comme nous l'avons déjà abordé plusieurs fois, il faut que cette notion de sécurité contienne des arguments qu'il sera possible de mesurer.

C'est pourquoi nous allons nous baser, pour notre analyse, sur deux données mesurables qui représenteront, pour nous, la notion de sécurité du logiciel que nous allons étudier[20] :

- Les failles publiées d'un logiciel
- Les exploits portés sur un logiciel

Nous ferons aussi la relation entre les exploits portés sur un logiciel par rapport aux vulnérabilités que celui-ci comporte.

Ces données mesurables ont pour avantage d'être générales et disponibles pour chaque logiciel existant. Elles ne sont donc pas spécifiques à certaines pratiques de programmation, certaines catégories de logiciels, certains langages de programmation, etc.

8.2. Choisir les logiciels à comparer

Pour pouvoir déterminer quelle est la catégorie la plus sécurisée entre les logiciels en source ouverte et en source fermée, il faut pouvoir analyser et comparer des données de software de

20 Bien évidemment, un logiciel sécurisé ne se définit pas uniquement par ces éléments. Mais, comme nous l'avons déjà abordé, il est presque impossible de tout analyser. Il faut donc, à un moment donné, faire des choix.

chacune de ces deux catégories. Nous allons donc maintenant choisir ces logiciels.

Comme il existe des dizaines de milliers de logiciels qui touchent plusieurs domaines de l'informatique, il est presque impossible de faire une analyse de tout ce qu'il existe. Nous allons donc suivre la même démarche que [Schryen & Kadura, 2009], [Schryen, 2011], [Alhazmi & Malaiya & &Ray, 2005] ou encore [Walden & Doyle & Welch & Whelan, 2009] : nous allons faire une sélection parmi tous ces domaines et tous ces logiciels.

Nous avons dès lors décidé de choisir, pour notre analyse, 3 branches de l'informatique :

- les systèmes d'exploitation
- les suites bureautiques
- les serveurs web

Le but de ce mémoire étant de faire une comparaison entre des logiciels open source et closed source, nous avons décidé de choisir, dans chaque branche, un logiciel développé en source ouverte ainsi qu'un logiciel développé en source fermée[21] :

- les systèmes d'exploitation
  - open source : le noyau Linux
  - closed source : Windows 7
- les suites bureautiques
  - open source : Libre Office
  - closed source : Microsoft Office
- les serveurs web
  - open source : Apache HTTP Serveur
  - closed source : IIS Serveur

Même si cela ne représentera pas l'entièreté de l'existant, nous aurons des résultats concernant ces logiciels nous permettant déjà de voir si une tendance en ressort.

8.3. Analyse objective

Pour réaliser une analyse objective, il faut utiliser des données concrètes et vérifiables. Mais il faut aussi, comme le disent [Schryen & Kadura, 2009] pouvoir faire abstraction de la personne, du rôle, de l'organisation gérant les différents logiciels choisis. Cela évite ainsi de prendre en compte des avis subjectifs ou encore d'avoir un avis ou une prise de position avant même d'avoir commencé.

21 Voir Annexe « 2. Le choix des logiciels »

C'est pourquoi nous ferons tout d'abord une analyse des logiciels en fonction de leurs branches de l'informatique.

Une fois cela fait, nous attribuerons sur les logiciels leur « étiquette » open source ou closed source pour comparer ces deux manières de développer.

9. Les failles publiées

9.1. La méthodologie

9.1.1. Le but de l'analyse

Le but de cette première analyse sera de montrer, chiffres à l'appui, s'il est possible de déterminer qui des logiciels open source ou closed source sont les plus sécurisés. Pour pouvoir démontrer cela, il faut avant tout définir les critères de ce que l'on appelle un logiciel sécurisé.

Plusieurs modèles de sécurité ont été établis suivant différents critères sélectionnés par les auteurs comme le patch d'index [Schryen & Kadura, 2009], le nombre de vulnérabilités patchées ou non [Schryen, 2011], les vulnérabilités en fonction de la densité du code [Alhazmi & Malaiya & Ray, 2005], etc.

En ce qui nous concerne, nous étudierons la sécurité d'un logiciel par rapport aux failles publiées. On analyse ainsi le logiciel tel qu'il est livré, tel qu'il est utilisé par les utilisateurs et tel qu'il peut être exploité par les hackers.

Ces failles reprennent l'ensemble des vulnérabilités qui ont été trouvées sur le logiciel et qu'il est ou a été possible d'exploiter à un moment donné de la vie du software. Cependant, cela ne représente pas l'ensemble des failles existantes car, comme le dit [Schryen, 2011], il est possible que des gestionnaires n'en publient pas.

9.1.2. Recueillir les données

Pour analyser les vulnérabilités sur les logiciels, nous allons nous baser sur le dictionnaire [CVE] de MITRE. [Schryen, 2011] nous informe que ce dictionnaire a pour avantage que toutes les vulnérabilités ont été inspectées par des professionnels de la sécurité. De plus, il est assez compréhensible.

Pour connaître la sévérité des failles, nous allons nous baser sur l'indice [CVSS] que nous allons arrondir au plus proche[22] pour avoir une notation sans virgule entre 1 et 10.

Pour récolter les données par rapport aux failles découvertes sur les logiciels que nous avons choisis, nous allons nous baser sur [CVE details][23].

Le but est de réaliser une comparaison sur plusieurs années. Nous allons, pour autant que cela soit possible, récupérer les données à partir de l'année 2015 et en finissant à l'année 2011.

9.1.3. Mettre un poids sur les vulnérabilités

Calculer le nombre de vulnérabilités ne suffit pas pour effectuer une analyse correcte de la sécurité d'un logiciel. Il serait très réducteur de dire : Plus un logiciel aura de failles, moins il est sécurisé. Moins un logiciel a de failles, plus il est sécurisé.

En effet, comme le font remarquer [Schryen & Kadura, 2009], une grosse faille, facile à trouver

22 Aussi appelé arrondi arithmétique.
23 Voir Annexe 3 « Lorsqu'il y a peu de données »

et exploiter, avec un fort impact sur le système n'aura pas les mêmes conséquences qu'une faille, difficile à trouver et exploiter, qui n'arrive qu'à de rares moments/conditions et qui n'a pas d'impacts sur le système. C'est pourquoi [Schryen & Kadura, 2009] utilisent un système de poids sur les vulnérabilités qui permet d'analyser ces failles en fonction de leur impact.

Nous allons nous baser sur ce même système : nous prendrons les failles détectées ainsi que leur « poids » qui ira de 1 à 10 : plus il est petit, moindre est l'impact. Plus il est grand, plus fort est l'impact.

Pour connaître le niveau de sécurité final d'un logiciel, nous additionnerons tous les poids des vulnérabilités trouvées, ce qui nous donnera la « non fiabilités » totale que nous pourrons comparer.

Par exemple :

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombres de failles	*8*	*5*	*2*	*4*	*5*	*2*	*0*	*1*	*2*	*1*

Logiciel 1

Le logiciel 1 a 30[24] vulnérabilités découvertes. Avec la répartition en poids, le logiciel a une valeur de 113[25].

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombres de failles	*1*	*2*	*5*	*2*	*1*	*2*	*4*	*4*	*4*	*5*

Logiciel 2

Le logiciel 2 a aussi un total de 30 vulnérabilités découvertes. Avec la répartition en poids, le logiciel a une valeur de 191.

En comparant nos deux exemples, qui ont le même nombre de failles découvertes mais que l'on a catégorisés en poids, on peut donc définir que le logiciel 2 est moins sécurisé que le logiciel 1 : son poids est plus grand : ses failles ont donc plus de conséquences.

9.1.4. Travailler par rapport à un pourcentage

Le nombre de vulnérabilités trouvées ne reflète pas non plus à quel point un logiciel est fiable ou non. Ce qui veut dire qu' un logiciel avec 25 vulnérabilités publiées n'est pas nécessairement plus fiable qu'un logiciel avec 200 vulnérabilités. [Hoepman & Jacobs, 2013] et [Schryen, 2011] expliquent cela en montrant que ce n'est pas parce que des failles sont découvertes qu'elles sont nécessairement publiées ou même corrigées. Ce qui fait que pour certains logiciels, on peut très bien se retrouver avec moins de vulnérabilités publiées qu'il n'y en a vraiment.

Pour minimiser l'erreur due à une comparaison faussée par ce nombre de vulnérabilités publiées et donc, pour éviter de comparer un logiciel dont toutes les failles sont publiées par rapport à un logiciel dont la plupart des failles ne sont pas rendues publiques, nous avons décidé, à partir des données connues, de faire une moyenne des failles par rapport à un pourcentage. Ainsi, à partir de

24 8+5+2+4+5+2+0+1+2+1

25 (8*1)+(5*2)+(2*3)+(4*4)+(5*5)+(2*6)+(0*7)+(1*8)+(2*9)+(1*10)

notre nombre de failles et de nombre poids, nous prendrons le poids total, nous le diviserons par le nombre de vulnérabilités, et nous multiplierons le tout par 100.

Par exemple :

Poids	1	2	3	4	5	6	7	8	9	10
Nombres de failles	*10*	*5*	*5*	*5*	*7*	*1*	*2*	*3*	*1*	*1*

Logiciel 1

Le logiciel 1 a 40 failles et le poids total est de 153. Ramené sur un pourcentage, le logiciel a un poids de 382,5[26] pour 100 vulnérabilités.

Poids	1	2	3	4	5	6	7	8	9	10
Nombres de failles	*2*	*1*	*2*	*2*	*1*	*1*	*3*	*1*	*4*	*3*

Logiciel 2

Le logiciel 2 a 20 failles et le poids total est de 153. Ramené sur un pourcentage, le logiciel a un poids de 620 pour 100 vulnérabilités.

En faisant une première analyse, on aurait pu se dire que le logiciel 2 est plus sûr car il a moins de failles pour un poids total identique. Mais lorsque l'on compare ces deux logiciels par rapport à 100 vulnérabilités, on remarque que le logiciel 2 est finalement beaucoup moins sûr et que son poids est beaucoup plus grand. Ses failles ont donc au final plus d'impacts que le logiciel 1.

9.2. Comparaison des failles entre logiciels de la même branche informatique

Par rapport aux données récoltées[27], nous allons comparer le poids des vulnérabilités des logiciels que nous avons choisis par rapport à leur branche.

9.2.1. Les systèmes d'exploitation

Nous allons comparer les données récoltées du noyau Linux[28] par rapport à Windows 7[29].

	2015	2014	2013	2012	2011	Total
Noyau Linux	*551*	*526*	*487*	*503*	*513*	2579
Windows7	*667*	*742*	*659*	*745*	*743*	3557
Différence	***117***	***216***	***172***	***243***	***230***	**978**

Tableau récapitulatif des failles du noyau Linux et de Windows 7

26 (153/40)*100
27 Voir Annexe 4 « Les totaux ne correspondent pas »
28 Voir Annexe 5 « Failles du noyau Linux »
29 Voir Annexe 6 « Failles de Windows 7 »

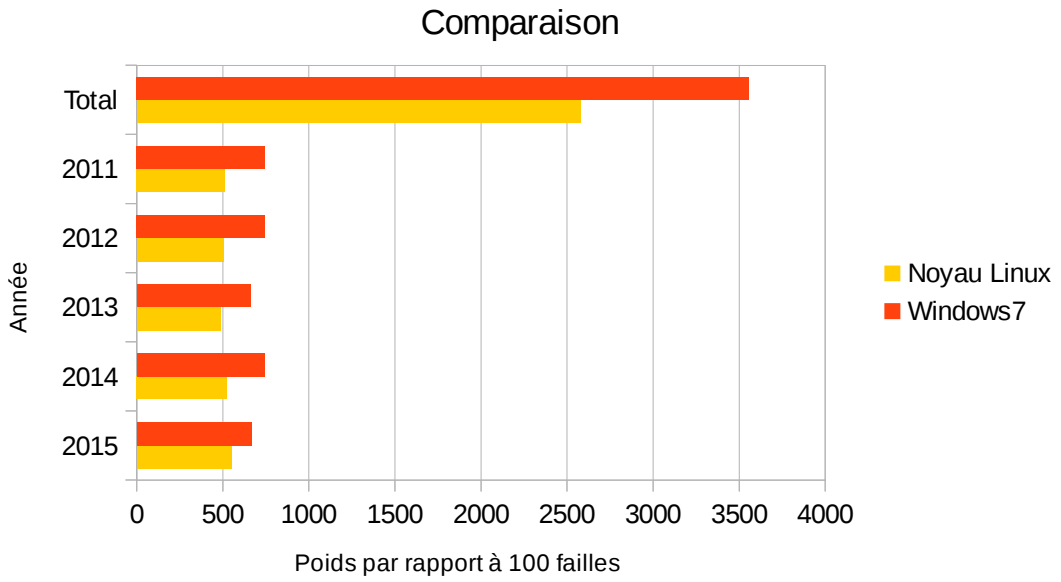


Diagramme de comparaison des failles entre le noyau Linux et Windows 7

Pour chacune des années étudiées, le poids des vulnérabilités sur 100 failles est toujours plus grand sur Windows 7 que sur le noyau Linux.

La différence minimum est de 117 sur l'année 2015 et la plus grande différence est sur l'année 2012 avec un écart de 243.

Le poids total de vulnérabilités est donc beaucoup plus important sur Windows 7 que sur le noyau Linux avec un écart de 978.

9.2.2. Les suites bureautiques

Nous allons maintenant comparer les données récoltées de Libre Office[30] à celles de Microsoft Office[31].

	2015	**2014**	**2013**	**2012**	**2011**	**Total**
Libre Office	*640*	*867*	*0*	*620*	*650*	2777
Microsoft Office	*863*	*790*	*782*	*853*	*900*	4187
Différence	***223***	***77***	***782***	***233***	***250***	**1411**

Tableau récapitulatif des failles de Libre Office et de Microsoft Office

30 Voir Annexe 7 « Failles de Libre Office »
31 Voir Annexe 8 « Failles de Microsoft Office »

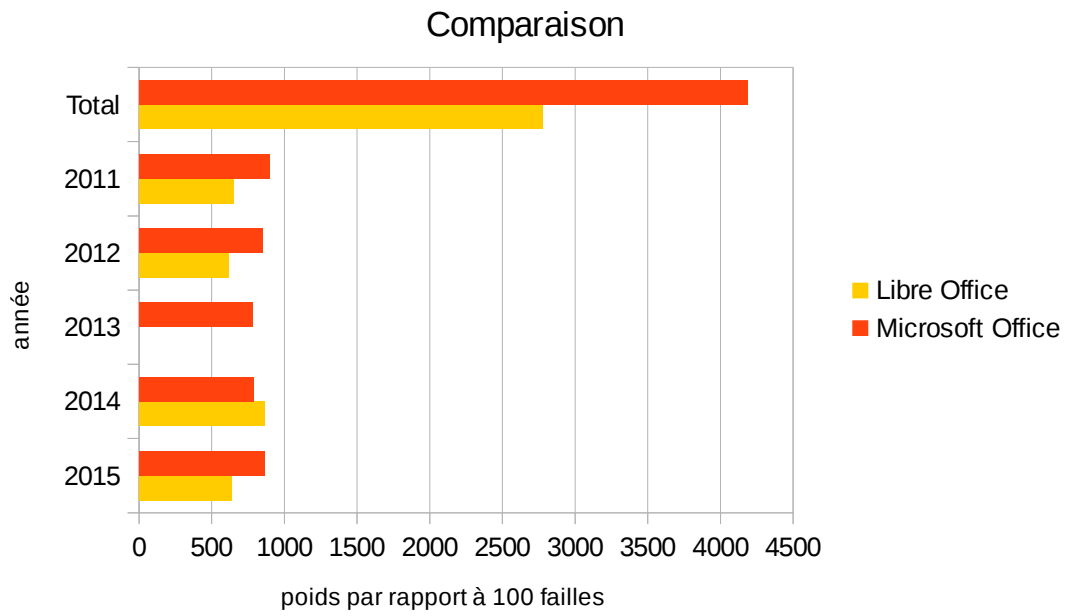


Diagramme de comparaison des failles entre Libre Office et Microsoft Office

À part l'année 2014, où le poids des vulnérabilités sur 100 failles est moindre sur Microsoft Office avec une différence de 77 par rapport à Libre Office, les autres années, c'est Libre Office qui a une moyenne de poids la moins grande.

L'écart minimum est de 223 pour l'année 2015 et l'écart le plus grand est de 782 pour l'année 2013. Il faut dire aussi que pour cette dernière année, aucune donnée n'a été récoltée concernant Libre Office alors que 17 failles ont été recensées sur Microsoft.

Le poids total de vulnérabilités est donc beaucoup plus important sur Microsoft Office que sur le Libre Office avec un écart de 1411. Même en ne tenant pas compte de l'année 2013, Microsoft Office aurait un poids supérieur avec une différence de 629.

9.2.3. Les serveurs web

Pour finir, nous allons comparer les données récoltées d'Apache HTTP Serveur[32] à celles d'IIS Serveur[33]. En ce qui concerne ces deux cas, très peu de données ont pu être récoltées. Nous avons donc décidé de procéder de manière différente : nous avons cumulé, dans un seul et même tableau, les données récoltées entre les années 2015 et 2008. Nous n'aurons plus de comparaison par année mais directement le total des données.

	Total
Apache Http Serveur	488
IIS Serveur	643
Différence	***154***

Tableau récapitulatif des failles d'Apache HTTP Serveur et d'IIS Serveur

32 Voir Annexe 9 « Failles d'Apache HTTP Serveur »
33 Voir Annexe 10 « Failles d'IIS Serveur »

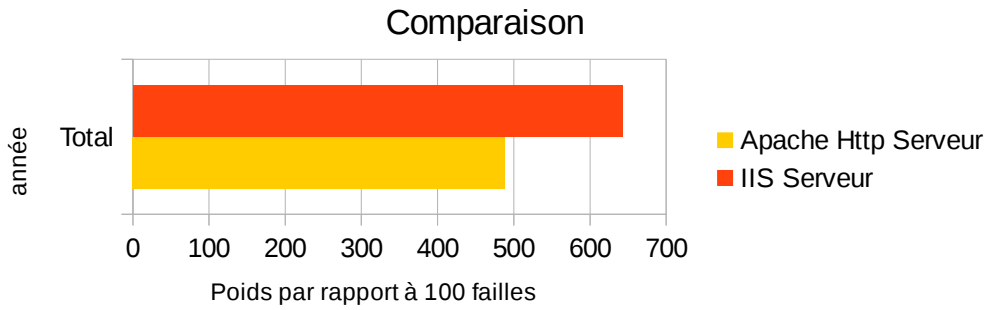


Diagramme de comparaison des failles entre Apache Http Serveur et IIS Serveur

On peut directement voir qu'Apache Http Serveur a un poids de vulnérabilités moindre qu'IIS Serveur avec une différence de 154.

Dans cette dernière analyse de logiciels, la moyenne des vulnérabilités sur 100 failles est donc plus importante sur IIS Serveur que sur Apache Http Serveur.

9.3. Comparaison des failles entre logiciels open source et closed source

Maintenant que nous avons effectué la comparaison entre logiciels, nous allons faire la comparaison totale entre logiciels open source et closed source. Nous allons additionner les données de chaque logiciel en source ouverte et en source fermée pour pouvoir comparer les données par rapport à ces deux types de développements.

	2015	2014	2013	2012	2011	Total
Noyau Linux	*551*	*526*	*487*	*503*	*513*	2579
Libre Office	*640*	*867*	*0*	*620*	*650*	*2777*
Apache Http						488
Open Source	***1191***	***1392***	***487***	***1123***	***1163***	**5844**
Windows 7	*667*	*742*	*659*	*745*	*743*	3557
Microsoft Office	*863*	*790*	*782*	*853*	*900*	*4187*
IIS Server						643
Closed Source	***1530***	***1532***	***1441***	***1598***	***1643***	**8387**
Différence	**339**	**139**	**955**	**475**	**480**	**2543**

Tableau récapitulatif des failles entre logiciels open source et closed source

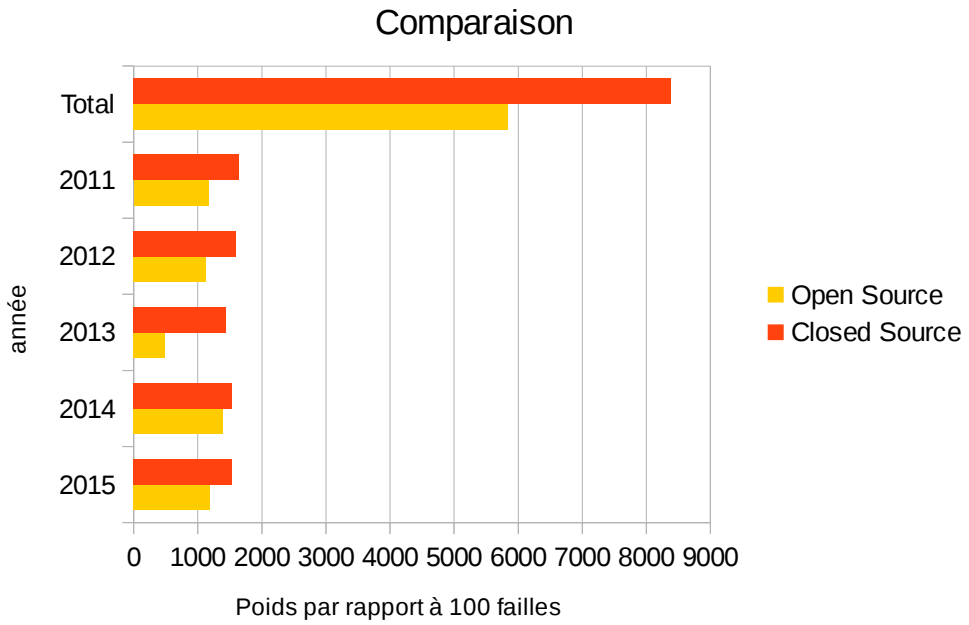


Diagramme de comparaison des failles entre logiciels open source et closed source

Étant donné que tous les précédents résultats étaient en faveur des logiciels open source, la mise en commun des résultats confirme cette tendance avec un poids de vulnérabilité moins grand pour chaque année étudiée.

L'écart le moins important est de 139 en 2014. Cette différence s'explique par le fait que cette année-là, Microsoft Office avait un moindre poids que Libre Office. Mais la différence est plus petite que celle entre Le Noyau Linux et Windows 7, ce qui fait que l'open source reste devant.

L'écart le plus grand est de 955 pour l'année 2013. À savoir que cela correspond à l'année où il n'y avait pas de données concernant Libre Office. Mais même sans cette donnée, l'open source aurait eu un poids plus faible vu que le Noyau Linux avait un moindre poids par rapport à Windows 7.

Le poids total de failles est donc en faveur de l'open source avec une différence de 2543.

Les résultats finaux de cette première analyse montrent que le poids des vulnérabilités sur 100 failles est plus important sur les logiciels closed source que sur les logiciels open source.

10. Les exploits de hackers

10.1. La méthodologie

10.1.1. Le but de l'analyse

Comme le montre [Seebruck, 2015], le terme de hacker est souvent utilisé pour désigner une personne mal intentionnée. Mais une classification[34] a été réalisée pour faciliter leurs identifications et motivations.

Dans cette deuxième analyse, nous allons nous intéresser à ces motivations et voir s'il est possible de déterminer si les hackers préfèrent attaquer des logiciels open source ou des logiciels closed source.

Nous allons donc essayer de mesurer cette donnée par rapport au nombre d'exploits qu'ils ont réussis à réaliser. Ces exploits représentent donc des vulnérabilités que des personnes ont réussis à utiliser.

10.1.2. Récolter les données

Pour récolter les données, nous allons analyser les différentes attaques qui ont été portées sur les logiciels étudiés et qui sont disponibles sur [Exploit DB]. Sur ce site, tout un chacun peut y envoyer ces exploits, c'est-à-dire les vulnérabilités qu'ils ont détectées et qu'ils ont réussi à exploiter.

Ces exploits peuvent ensuite être vérifiés et donc validés par [Offensive Security].

Nous allons de nouveau récolter les données sur plusieurs années, en commençant par l'année 2015 et en finissant par l'année 2010.

10.1.3. Le nombre d'exploits

Par rapport aux données récoltées, nous analyserons le nombre d'exploits portés sur les logiciels.

Nous déduirons d'un logiciel qui a le plus d'exploits reportés qu'il est le plus attrayant pour les hackers.

10.1.4. Le pourcentage de vérification[35]

Nous nous intéresserons aussi au pourcentage de vérifications effectuées par [Offensive Security] sur les logiciels. Ainsi, nous pourrons déterminer s'il y a plus d'attention portée sur les logiciels open source ou closed source.

10.2. Comparaison des exploits entre logiciels de la même branche informatique

Tout comme la partie précédente, nous allons tout d'abord comparer le poids des vulnérabilités des logiciels que nous avons choisis par rapport à leur branche.

34 En fonction de leurs compétences, motivations, expériences, etc
35 Voir Annexe 11 « Les exploits vérifiés »

10.2.1. Les systèmes d'exploitation

Nous comparons dans cette partie les données récoltées sur le noyau Linux[36] ainsi que sur Windows 7[37]

	Postés	Vérifiés	Total	% Verif.
Noyau Linux	*23*	*48*	*71*	*67.61%*
Windows 7	*4*	*38*	*42*	*90.48%*

Tableau récapitulatif des exploits portés sur le noyau Linux et sur Windows 7

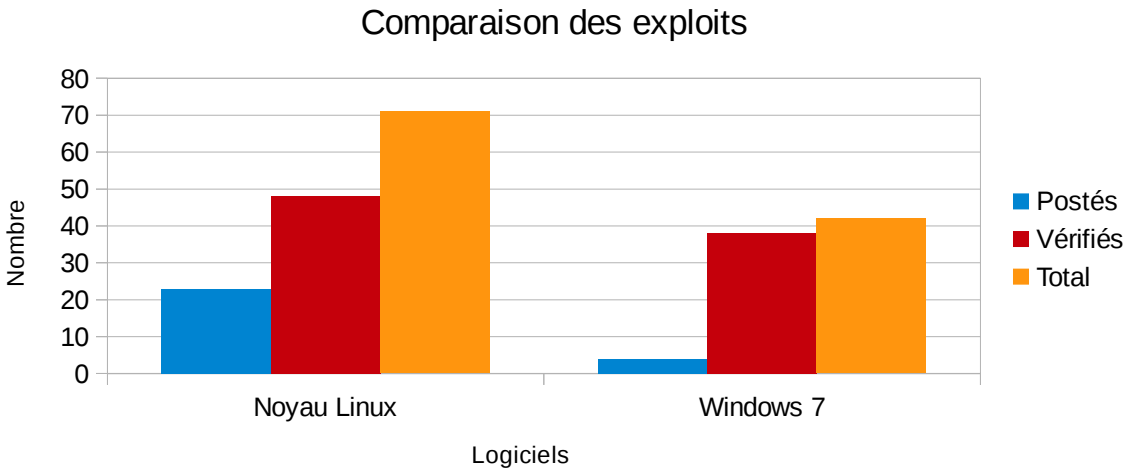


Diagramme de comparaison des exploits entre le noyau Linux et Windows 7

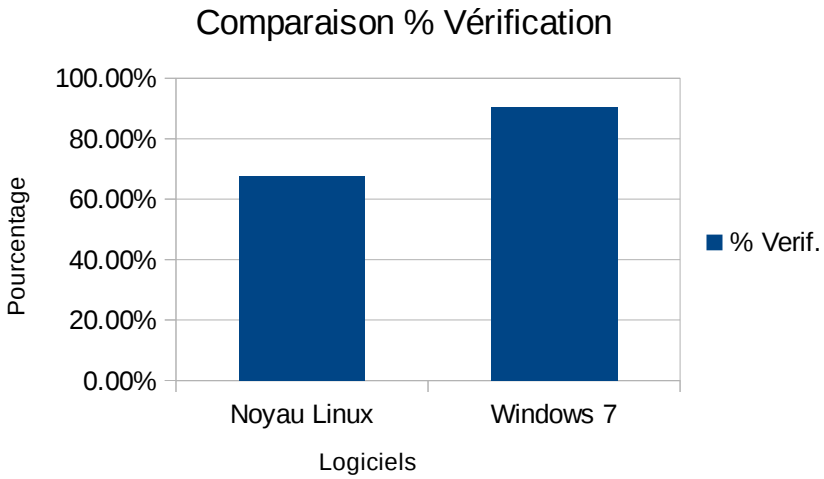


Diagramme de comparaison au niveau du pourcentage des vérifications des exploits entre le noyau Linux et Windows 7

Dans cette comparaison, nous pouvons remarquer qu'il y a plus d'exploits publiés et vérifiés sur le noyau Linux (71 au total) par rapport à Windows 7 (42 au total).

Par contre, le pourcentage de vérification des exploits montre qu'ils sont plus souvent vérifiés sur Windows 7 (avec plus de 90%) comparé au noyau Linux (avec un peu plus de 67%).

36 Voir Annexe 12 « Exploits sur le noyau Linux »
37 Voir Annexe 13 « Exploits sur Windows 7 »

10.2.2. Les suites bureautiques

Nous allons maintenant comparer les exploits réalisés sur Libre Office[38] et sur Microsoft Office[39].

	Postés	Vérifiés	Total	% Verif.
Libre Office	*1*	*1*	*2*	*50.00%*
Microsoft Office	*5*	*38*	*43*	*88.37%*

Tableau récapitulatif des exploits portés sur Libre Office et Microsoft Office

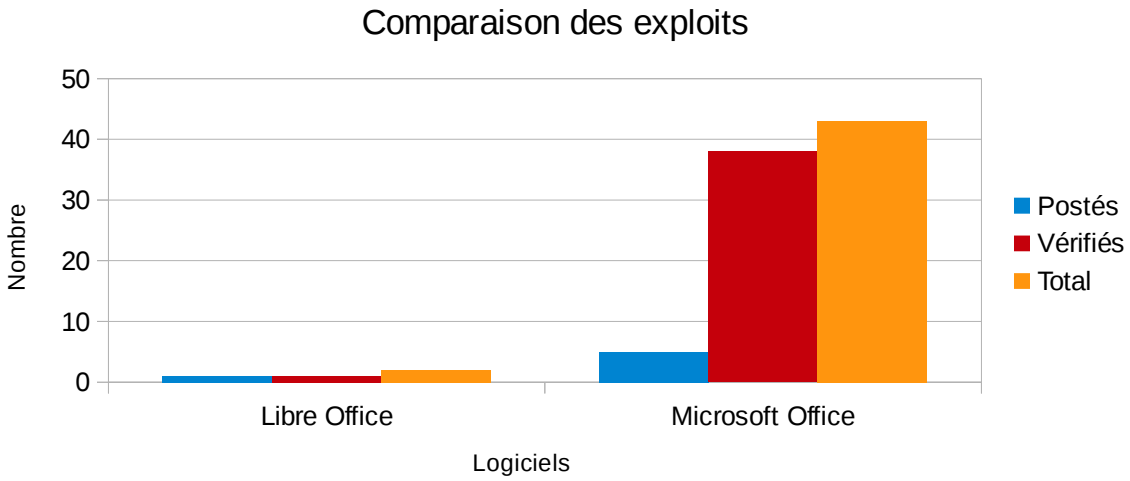


Diagramme de comparaison des exploits entre Libre Office et Microsoft Office

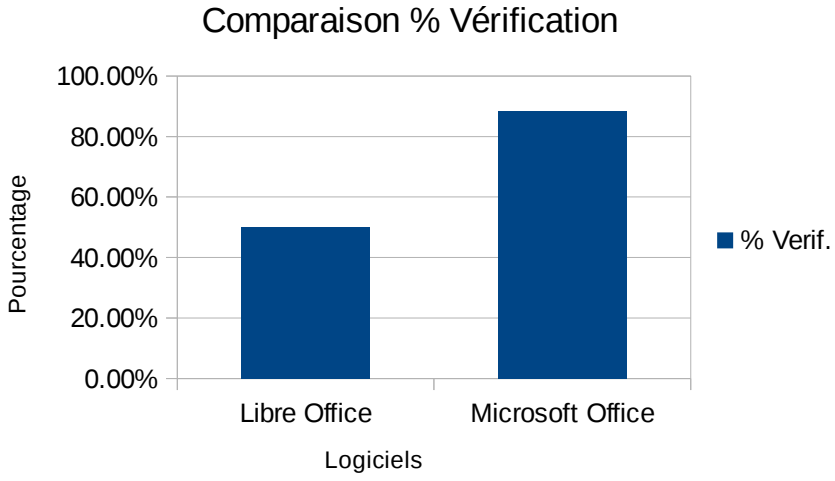


Diagramme de comparaison au niveau du pourcentage des vérifications des exploits entre Libre Office et Microsoft Office

Les résultats de cette comparaison montrent que Microsoft Office a beaucoup plus d'exploits réalisés (43) alors que seulement 2 ont été recensés sur le site pour Libre Office.

Alors qu'un des deux exploits a été vérifié pour Libre Office, menant le pourcentage de vérification à 50 %, plus de 88 % des 43 exploits de Microsoft Office ont été vérifiés.

38 Voir Annexe 14 « Exploits sur Libre Office »
39 Voir Annexe 15 « Exploits sur Microsoft Office »

10.2.3. Les serveurs web

Nous terminons donc en comparant les données disponibles sur Apache HTTP Serveur[40] et sur IIS Serveur[41].

	Postés	Vérifiés	Total	% Verif.
Apache Http Serveur	*2*	*4*	*6*	*66.67%*
Serveur IIS	*4*	*20*	*24*	*83.33%*

Tableau récapitulatif des exploits portés sur Apache HTTP serveur et sur le serveur IIS

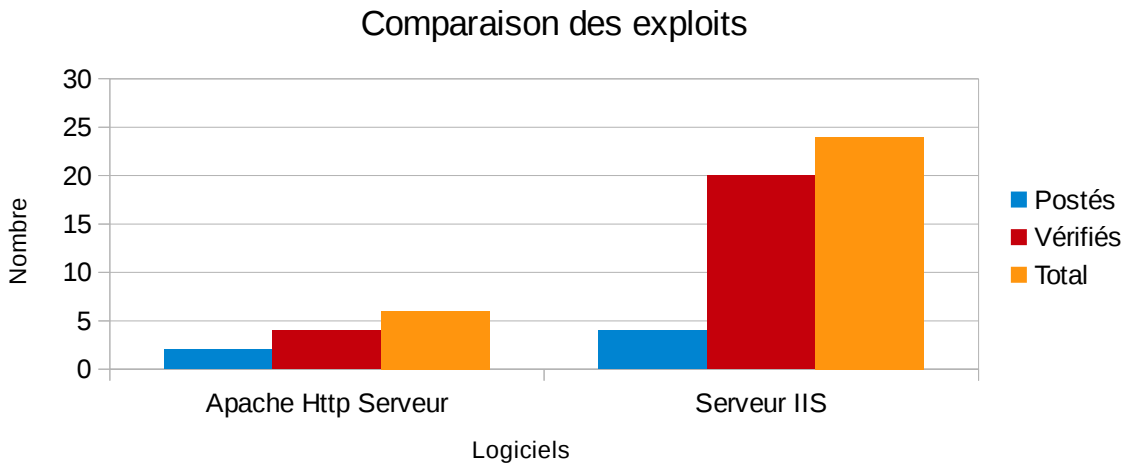


Diagramme de comparaison des exploits entre Apache HTTP serveur et le serveur IIS

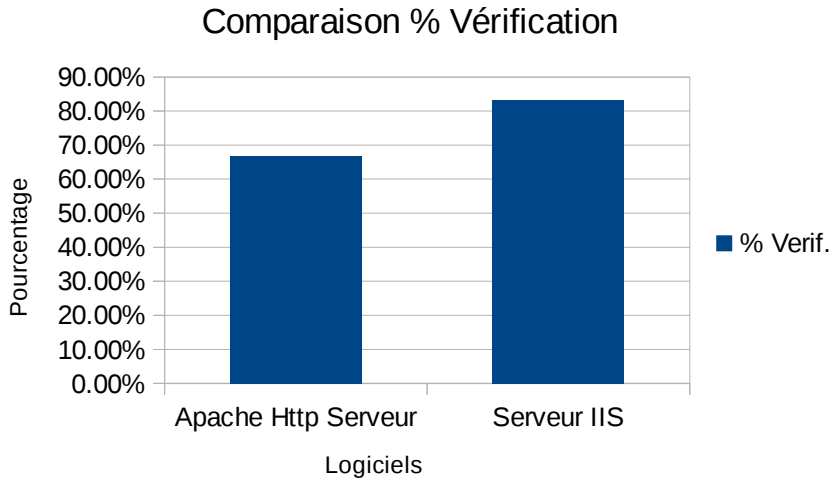


Diagramme de comparaison au niveau du pourcentage des vérifications des exploits entre le noyau Linux et Windows 7

Cette dernière comparaison montre qu'il y a peu d'exploits réalisés sur Apache (6) alors que quatre fois plus ont été portés sur le serveur IIS (24).

Il y a plus de vérifications qui ont été faites sur le serveur IIS avec un pourcentage de plus de 83 % alors que deux tiers des 6 exploits d'Apache (plus de 66 %) ont été vérifiés.

40 Voir Annexe 16 « Exploits sur Apache HTTP Serveur »
41 Voir Annexe 17 « Exploits sur IIS Serveur »

10.3. Comparaison des failles entre logiciels open source et closed source

De nouveau, tout comme la partie précédente, après avoir effectué la comparaison entre logiciels[42], nous réaliserons la comparaison entre logiciels open source et closed source en additionnant les résultats des logiciels de ces deux catégories.

	Postés	Vérifiés	Total	% Verif.
Noyau Linux	*23*	*48*	*71*	*67.61%*
Libre Office	*1*	*1*	*2*	*50.00%*
Apache Http Serveur	*2*	*4*	*6*	*66.67%*
Open Source	***26***	***53***	***79***	***67.09%***
Windows 7	*4*	*38*	*42*	*90.48%*
Serveur IIS	*4*	*20*	*24*	*83.33%*
Microsoft Office	*5*	*38*	*43*	*88.37%*
Closed Source	***13***	***96***	***109***	***88.07%***

Tableau récapitulatif des failles entre logiciels open source et closed source

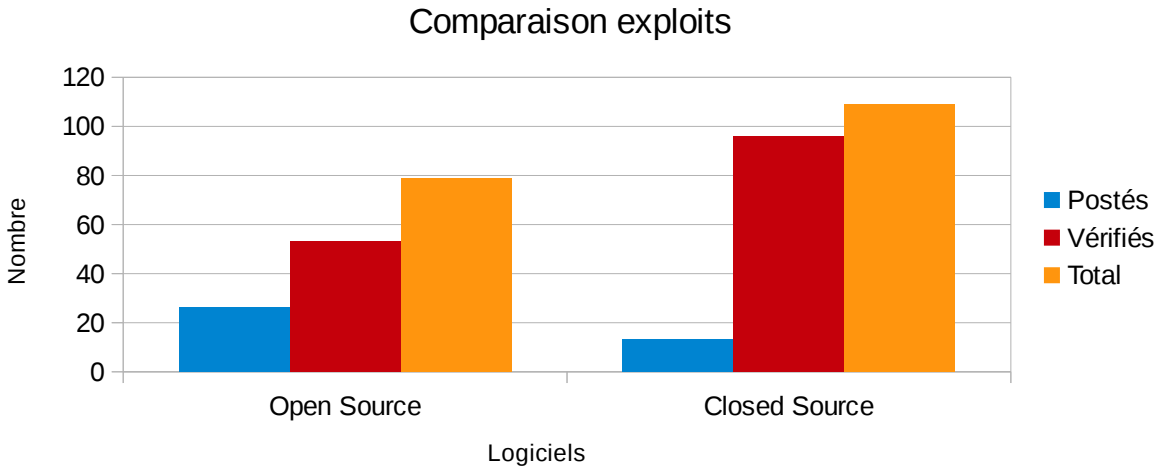


Diagramme de comparaison des exploits entre logiciels open source et closed source

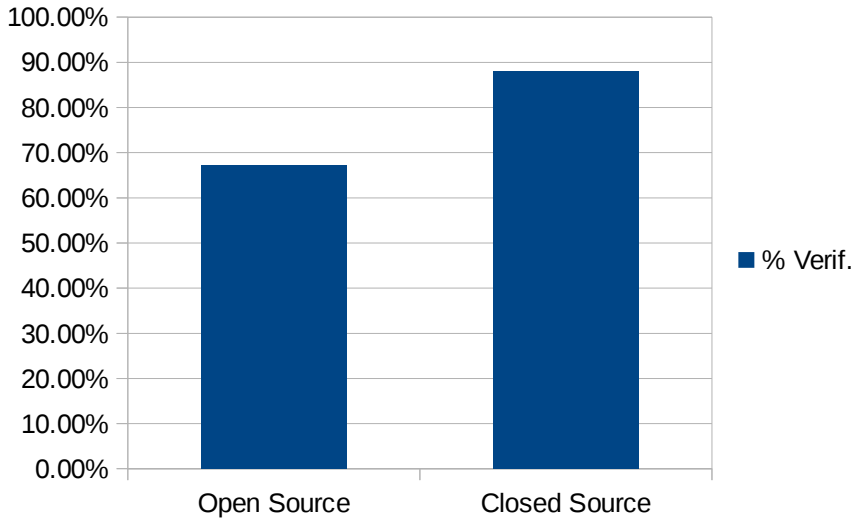


Diagramme de comparaison au niveau du pourcentage des vérifications des exploits entre logiciels open source et closed source

42 Voir Annexe 18 « Comparaison des exploits entre les 6 logiciels »

En comparant les systèmes d'exploitation, le noyau Linux avait le plus de failles exploitées comparés à Windows 7. En faisant la comparaison entre tous les logiciels, c'est même le logiciel qui a le plus de failles exploitées avec 71 exploits. Le deuxième logiciel avec le plus d'exploits est Microsoft Office avec 43 exploits, soit 28 en moins que le noyau Linux. Cependant, le nombre de failles exploitées des deux autres logiciels open source étudiés est tellement bas comparé aux logiciels closed source qu'au final, il y a moins de failles exploitées sur les logiciels en source ouverte avec 79 exploits par rapport aux logiciels en source fermée qui ont 109 failles exploitées.

En comparant les logiciels indépendamment, pour chaque comparaison, les logiciels open source avaient un pourcentage de vérification d'exploits moindre par rapport aux logiciels closed source. Il est donc normal que le regroupement que nous avons effectué confirme cette tendance avec plus de 88 % de vérifications faites sur les exploits des logiciels en source fermée par rapport à plus de 67 % de vérification sur les exploits de logiciels en source ouverte. On peut même remarquer que, même s'il y a moins de failles exploitées sur les logiciels open source, le nombre d'exploits postés mais non vérifiés, qui est de 26, est plus grand que le nombre d'exploits postés et non vérifiés des logiciels closed source où le nombre est seulement de 13. Ce qui confirme bien que les exploits de logiciels en source fermée sont plus souvent vérifiés que les exploits de logiciels en source ouverte.

De ce que nous venons d'analyser, les chiffres des logiciels open source sont toujours plus bas, que ce soit par rapport aux exploits réalisés ou par rapport au pourcentage de vérification des exploits, on peut donc déduire que les logiciels closed source attirent plus les hackers.

11. Les failles publiées par rapport aux exploits réalisés

11.1. La méthodologie

11.1.1. Le but de l'analyse

Après avoir comparé les failles et ensuite les exploits au niveau des logiciels, nous allons, dans cette troisième et dernière analyse, mettre en commun les données entre les deux premières analyses.

Nous allons donc faire une comparaison entre les exploits et les failles des logiciels et voir, entre autres, qui, entre les logiciels open source et closed source a les failles, par rapport à la liste de la première analyse, les plus souvent exploitées.

11.1.2. Récolter les données

Pour récolter les données, nous utiliserons de nouveau les sites utilisés lors des deux premières analyses. C'est-à-dire [CVE details] pour récolter les attaques faites sur les logiciels et [Exploit DB] pour récolter les informations sur les exploits qui ont été réalisés.

Nous effectuerons une analyse allant de l'année 2015 à l'année 2010.

11.1.3. Mettre un poids sur les exploits

En reprenant la même méthodologie que la première partie, nous allons mettre un poids sur les exploits qui ont été réalisés. Nous utiliserons donc utiliser le dictionnaire [CVE] pour définir les exploits et l'indice [CVSS] arrondi au plus proche pour connaître la sévérité de la faille. Et, de nouveau, nous allons travailler par rapport à une moyenne à partir d'un pourcentage pour effectuer la comparaison entre les logiciels.

Notons que tous les exploits n'ont pas de numéro [CVE] d'identifié. Il n'est donc pas possible de connaître l'indice [CVSS] et donc, d'évaluer le poids de ces exploits en question. Nous ne nous occuperons pas de ces cas-là.

11.1.4. Comparaison du pourcentage d'exploits par rapport à la liste des failles

Nous possédons donc une liste de failles qui ont été publiées et une liste d'exploits qui ont été réalisés. Nous réaliserons dès lors une comparaison entre ces différentes données pour évaluer le pourcentage d'exploits en fonction du nombre de failles.

Pour cela, nous avons effectué un récapitulatif des exploits et nous les avons marqués comme étant listés dans la liste récupérée via [CVE details] ou non. À savoir que ce n'est pas parce qu'un exploit est classé comme « Non listé » que cela signifie qu'il ne se retrouve pas sur le site de [CVE details]. Au contraire : tous les exploits réalisés étaient associés à une faille publiée. Le fait que des exploits soient classés en « non listés » correspond au fait qu'[Exploit DB] et [CVE details] peuvent classer la même donnée de manière différente.

Par exemple :

– Un exploit sur [Exploit DB] peut être défini comme étant effectué sur le logiciel « Microsoft Office ». Nous comptabilisons donc un exploit en plus. Mais la faille listée correspondante peut être marquée sur [CVE details] comme affectant le logiciel « Microsoft Word » et non « Microsoft Office » et donc, ne faisant pas partie de la liste des failles que nous avons récupérée de « Microsoft Office ». Nous marquons donc, dans ce cas-là, l'exploit comme non listé.

Le fait de distinguer les exploits listés et non listés va nous permettre d'avoir un nombre total de failles correct pour obtenir notre pourcentage. Cela, en additionnant simplement le nombre de failles totales de la liste de [CVE details] avec le nombre d'exploits que nous avons marqués comme non listés.

Par exemple, nous allons prendre 100 failles listées, avec 50 exploits réalisés mais dont 40 non listés :

– En ne tenant pas compte des exploits non listés, et donc, en comparant des exploits qui ne se trouvent pas dans la liste des 100 failles listées, nous obtenons un pourcentage de (100/100)*50. Soit, 50 % d'exploits réalisés sur les failles du logiciel.

– En comptabilisant les exploits non listés à la liste des 100 failles, nous obtenons un total de failles de 140, ce qui change le calcul en : (100/140)*50. Soit, 35,71 % d'exploits réalisés sur l'ensemble des failles que nous avons recensées sur le logiciel.

11.2. Comparaison des exploits entre logiciels de la même branche informatique

Toujours en adoptant la même démarche que précédemment, nous allons tout d'abord réaliser une comparaison entre les logiciels de la même branche informatique.

11.2.1. Les systèmes d'exploitation

Nous allons faire la comparaison entre le noyau Linux[43] et Windows 7[44]

	Noyau Linux	**Windows 7**	**Différence**
Poids/100 exploits	*570*	*752*	*182*

43 Voir Annexe 19 « Failles et exploits du noyau Linux »
44 Voir Annexe 20 « Failles et exploits de Windows 7 »

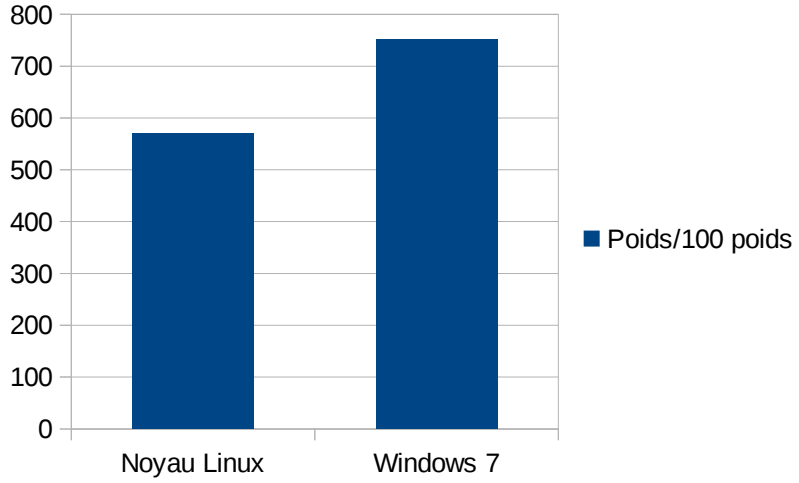
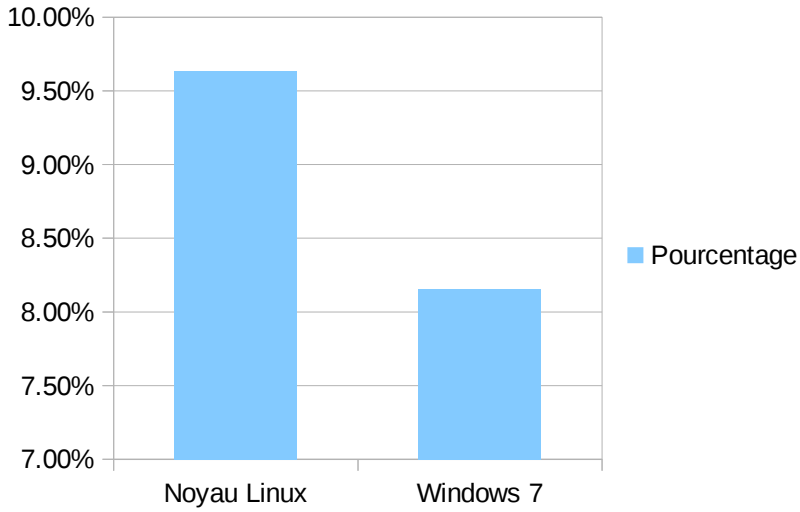


Diagramme de comparaison du poids des exploits entre le noyau Linux et Windows 7

En ce qui concerne le noyau Linux et Windows 7, le poids des failles listées sur le noyau Linux était moins important que sur Windows 7 mais cependant, il y avait plus d'exploits réalisés sur le noyau Linux.

En attribuant un poids sur les exploits, nous remarquons que le poids est moins important sur le noyau Linux avec une différence de poids de près de 182. Ce qui montre que, même si le système a plus d'exploits menés contre lui, la moyenne de leur poids est inférieure à celle de Windows 7. Cela peut s'expliquer par le fait que les failles disponibles sur le noyau Linux ont un poids inférieur au système de Microsoft.

	Noyau Linux	**Windows 7**
Pourcentage	*9.63%*	*8.16%*



Pourcentage d'exploits réalisés par rapport aux nombres de failles listées entre le noyau Linux et Windows 7[45]

Le nombre d'exploits réalisés, tout comme le nombre de failles total, est plus grand sur le noyau Linux que sur Windows 7 avec respectivement 71 exploits contre 42 et 737 failles contre 515. Cela ne signifie pas pour autant que le pourcentage d'exploits était plus important sur le noyau Linux que sur Windows 7. Mais ce fut cependant bien le cas avec un pourcentage de 9,63 % pour le noyau Linux par rapport à 8,16 % pour Windows 7.

45 Voir Annexe 21 « Récapitulatif des données de comparaison entre le noyau Linux et Windows 7 »

11.2.2. Les suites bureautiques

Cette comparaison se porte sur Libre Office[46] et Microsoft Office[47]

	Libre Office	Microsoft Office	Différence
Poids/100 poids	*0*	*823*	*823*

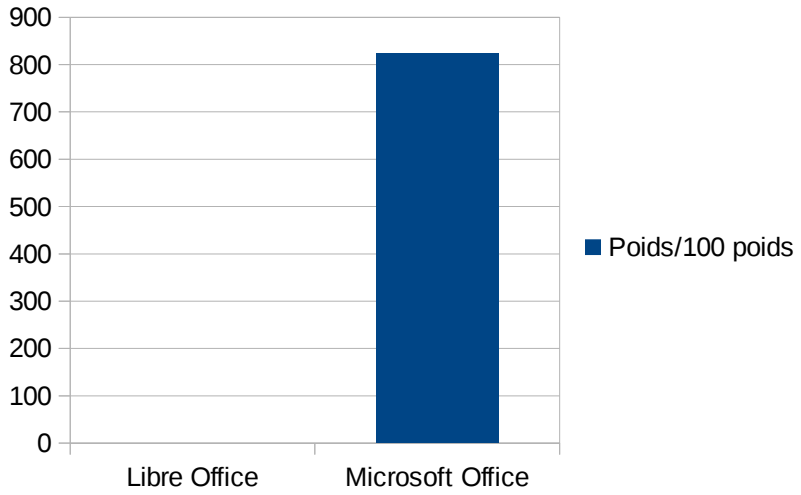
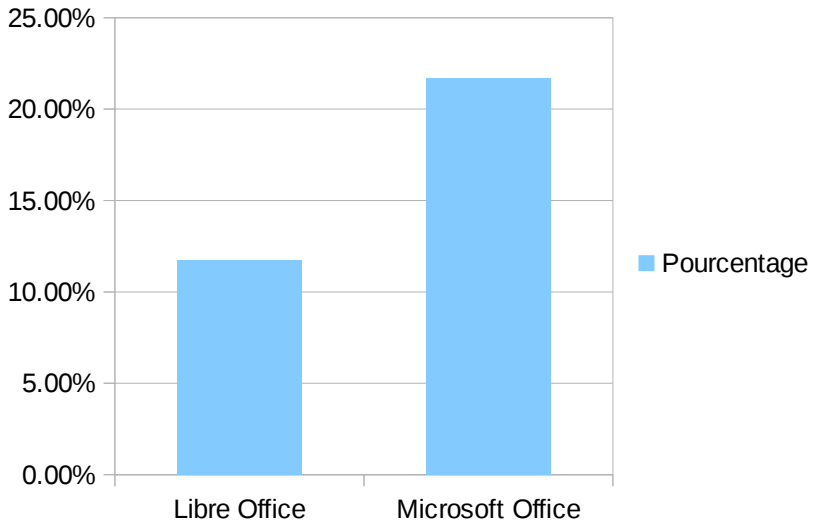


Diagramme de comparaison du poids des exploits entre Libre Office et Microsoft Office

En ce qui concerne Libre Office, il n'y avait déjà que 2 exploits recensés sur [Exploit DB] comparés à 43 exploits pour Microsoft Office. De plus, ces deux exploits ne sont liés à aucun numéro [CVE] et n'ont donc pas d'indice [CVSS]. Il n'est dès lors pas possible de retenir un poids pour les exploits Libre Office.

C'est dès lors normal que la moyenne du poids des exploits sur Microsoft Office, qui a 30 exploits dont nous avons pu établir un indice [CVSS], soit plus grande que Libre Office.

	Libre Office	Microsoft Office
Pourcentage	*11.76%*	*21.72%*



Pourcentage d'exploits réalisés par rapport au nombre de failles listées entre le noyau Linux et Windows 7[48]

46 Voir Annexe 22 « Failles et exploits de Libre Office »
47 Voir Annexe 23 « Failles et exploits de Microsoft Office »
48 Voir Annexe 24 « Récapitulatif des données de comparaison entre Libre Office et Microsoft Office »

Microsoft Office a 198 failles et 43 exploits à son actif. Ces données sont beaucoup plus grandes que celles de Libre office qui n'a que 17 failles et 2 exploits recensés. Et lorsque l'on compare le pourcentage d'exploits par rapport aux failles, celui-ci est aussi beaucoup plus grand sur le logiciel Windows avec plus de 21 % alors que Libre Office a un peu moins de 12 %.

11.2.3. Les serveurs web

Nous terminerons par comparer Apache HTTP Serveur[49] et IIS Serveur[50]

	Apache HTTP	IIS Serveur	Différence
Poids/100 poids	*633*	*806*	*173*

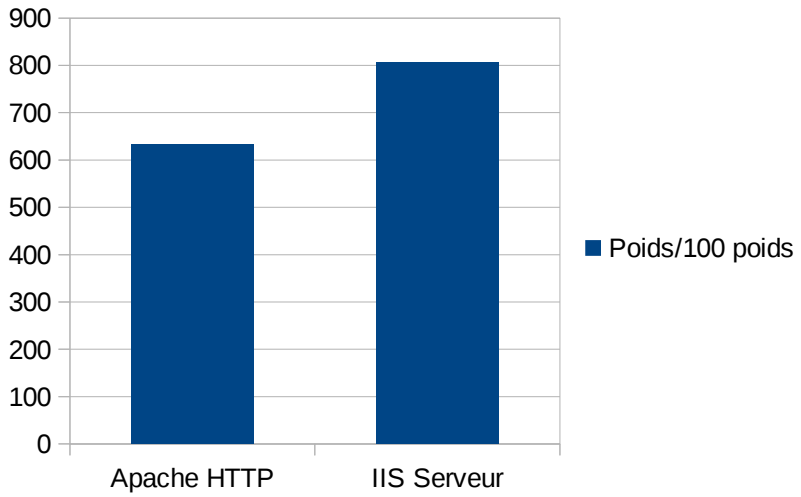
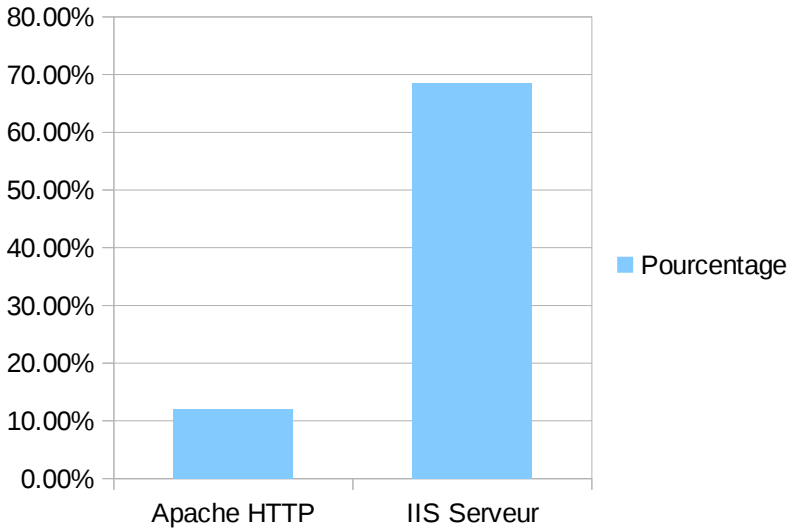


Diagramme de comparaison du poids des exploits entre Apache HTTP Serveur et IIS Serveur

Lors de l'analyse sur les failles listées, Apache HTTP Serveur avait une moyenne de poids plus faible que IIS Serveur. Cela se confirme aussi avec cette analyse du poids des exploits, où le logiciel apache possède un poids plus faible que son homologue avec une différence de poids de près de 173.

	Apache HTTP	IIS Serveur
Pourcentage	*12.00%*	*68.57%*

49 Voir Annexe 25 « Failles et exploits d'Apache HTTP Serveur »
50 Voir Annexe 26 « Failles et exploits de IIS Serveur »



Pourcentage d'exploits réalisés par rapport au nombre de failles listées entre le noyau Linux et Windows 7[51]

Alors qu'il y a moins d'exploits réalisés sur Apache HTTP Serveur, avec 6 exploits seulement face à 24 exploits, le total du nombre de failles était plus important sur Apache avec 50 exploits contre 35 sur IIS Serveur. C'est donc ce qui explique que le pourcentage d'exploits comparés aux nombres de failles soit si élevé sur IIS Serveur que sur Apache HTTP.

Cependant, pour nuancer un peu ces résultats, IIS Serveur est le logiciel pour lequel il y avait le plus d'exploits recensés sur [Exploit DB] mais non listés sur [CVE details]. Ainsi, seulement 3 exploits étaient listés contre 21 non listés, soit 87, 5 % d'exploits non listés. Ce qui fait que sur 35 failles, seulement 14 viennent de [CVE details] et 21 viennent d'[Exploit DB]. Il est donc normal que le pourcentage d'exploits comparés au nombre de failles soit si élevé pour IIS Serveur.

11.3. Comparaison des failles entre logiciels open source et closed source

Nous terminons cette série d'analyses en effectuant la mise en commun des données des logiciels du point de vue open source et closed source[52].

	Open Source	Closed Source	Différence
Poids/100 poids	*576*	*793*	*217*

51 Voir Annexe 27 « Récapitulatif des données de comparaison entre Apache HTTP Serveur et IIS Serveur »
52 Voir Annexe 28 « Récapitulatif des données de comparaison entre les logiciels open source et closed source »

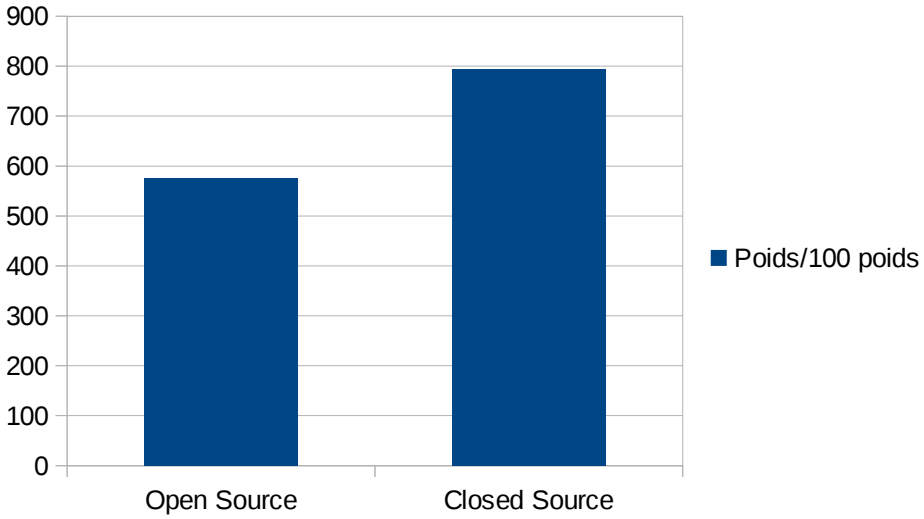
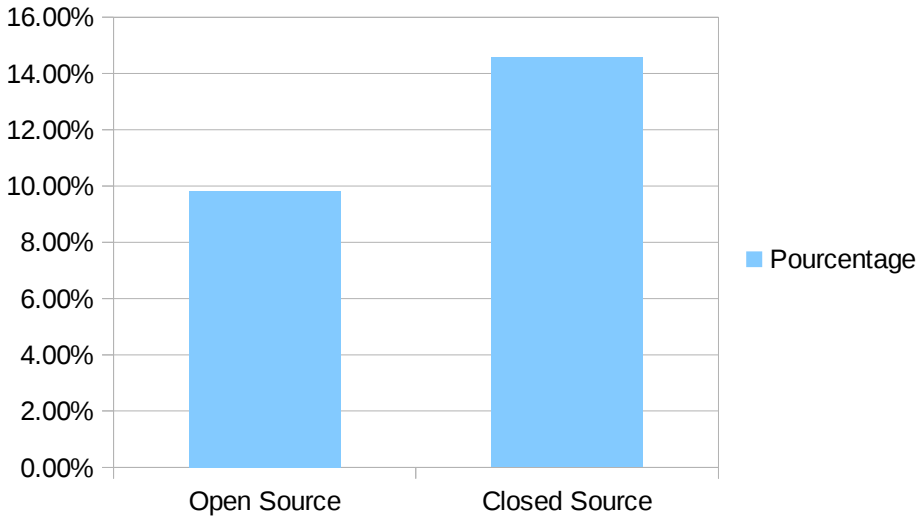


Diagramme de comparaison du poids des exploits entre les logiciels open source et closed source

Rien de bien surprenant en ce qui concerne ces chiffres : lors des analyses au niveau des logiciels en fonction de la branche informatique, à chaque fois, le logiciel open source avait une moyenne de poids plus faible que le logiciel closed source. Il est donc normal que la mise en commun montre que les exploits des logiciels en source ouverte aient une moyenne de poids plus léger, avec une différence de près de 217, par rapport aux exploits des logiciels en source fermée.

	Open Source	**Closed Source**
Pourcentage	*9.83%*	*14.57%*



Pourcentage d'exploits réalisés par rapport au nombre de failles listées entre le noyau Linux et Windows 7

Ici, en reprenant la comparaison au niveau des logiciels en fonction de la branche informatique, seul le noyau Linux, logiciel open source, avait un pourcentage plus élevé que son homologue. Pour les deux autres comparaisons, les logiciels closed source, Microsoft Office et IIS Serveur, avaient à chaque fois un pourcentage plus élevé.

En faisant la mise en commun, on peut voir que le nombre de failles total pour les logiciels en source ouverte est plus grand, avec 804 failles, que les logiciels en source fermée, avec 748 failles mais qu'il y a moins d'exploits portés sur les logiciels open source, avec 79 exploits, contre 109 exploits concernant les logiciels closed source. Le pourcentage d'exploits par rapport aux failles est donc plus important pour les logiciels en source fermée, avec 14,57 % par rapport aux logiciels en source ouverte qui ont un peu moins de 10 %.

12. Analyse des résultats

12.1. Les résultats

Notre recherche s'est basée sur 3 analyses :

– Une analyse du poids des failles sur une moyenne de 100 failles, où les logiciels open source avaient un poids de failles plus faible que les logiciels closed source.

– Une analyse du nombre d'exploits réalisés par les hackers ainsi que du pourcentage de vérification de ces exploits faite par [Offensive Security]. Les résultats montrent qu'il y a plus d'exploits portés sur les logiciels en source fermée, et qu'il y avait moins de vérifications faites sur les logiciels en source ouverte.

– Une analyse du poids des exploits ainsi que le pourcentage d'exploits réalisés par rapport au nombre de failles listées. Nous avons vu que le poids des exploits était moindre sur les logiciels open source et que le pourcentage d'exploits par rapport au nombre de failles était supérieur sur les logiciels closed source.

Autrement dit, pour chacune de ces analyses, à chaque fois, les logiciels open source récoltent de meilleurs résultats que les logiciels closed source. Ce qui amènerait à penser que les logiciels en source ouverte seraient plus sécurisés que les logiciels en source fermée.

12.2. Analyse

12.2.1. Lien entre le poids des failles et le poids des exploits

Avec les données récoltées, nous pouvons aussi remarquer un lien entre le poids des failles listées et le poids des exploits réalisés.

En effet, voici, pour rappel, le tableau de la moyenne du poids des attaques :

	Open Source	Closed Source
Poids/100 poids	*576*	*793*

Maintenant, nous allons faire la moyenne du poids des failles entre les logiciels open source et closed source. Nous obtenons le tableau ci-dessous[53] :

	Total	Moyenne
Noyau Linux	*2579*	*516*
Libre Office	*2777*	*555*
Apache Http	*488*	*488*
Open Source	***5844***	***520***
Windows 7	*3557*	*711*
Microsoft Office	*4187*	*837*
IIS Server	*643*	*643*
Closed Source	**8387**	**731**

Nous remarquons que les résultats entre le poids des exploits et le poids des failles sont relativement similaires (entre 500 et 600 pour l'open source et entre 700 et 800 pour le closed

53 Pour rappel, le poids des failles d'Apache HTTP serveur et du serveur IIS est déjà une moyenne sur les différentes années

source).

Cela permet donc de prouver ce que [Schryen & Kadura, 2009] avancent dans leur article : plus les failles sont importantes, plus les attaques pourront être néfastes au logiciel et aux utilisateurs qui l'utilisent.

12.2.2. Les motivations

[Seebruck, 2015] et [Floyd & Harrington & Hivale, 2007] montrent que les hackers ont différentes motivations. Dans ces motivations, il y a le prestige qui peut s'acquérir en fonction de la difficulté de l'exploit réalisé.

Comme le montrent [Hoepman & Jacobs, 2013], avec le closed source, le fait de ne pas pouvoir analyser le code peut rendre les choses plus compliquées. Les hackers ont plus de mal à accéder à l'information, à lancer une attaque. Mais cela rend les exploits plus grands s'ils arrivent à en réaliser.

Concernant l'open source, comme le montrent [Hoepman & Jacobs, 2013], le code est disponible pour inspection, modification. Il est donc plus facile de pouvoir y trouver et exploiter une faille.

On peut donc affirmer que les hackers ont plus de prestige et de notoriété à réussir à hacker une faille d'un logiciel closed source.

Une autre motivation avancée par [Floyd & Harrington & Hivale, 2007] est la curiosité. Par exemple, il est possible d'avoir envie de hacker un logiciel pour savoir comment il est réalisé. En ce qui concerne les logiciels open source, comme le code est mis à disposition, il n'est pas nécessaire de hacker quoi que ce soit dans ce but précis.

En partant de ces motivations, cela peut alors expliquer pourquoi nous avons récolté plus d'exploits réalisés contre les logiciels closed source ainsi qu'un plus grand pourcentage de failles qui ont permis de faire un exploit.

12.3. D'autres documents amenant au même résultat

Notre analyse portait principalement sur les données de failles listées et d'exploits réalisés. D'autres analyses ont été faites sur d'autres composants de la sécurité qui tendent vers le même résultat :

- [Schryen & Kadura, 2009], qui ont fait une analyse sur le patch d'index et le patch time. Cependant, comme ils le disent, cette analyse ne reflète pas vraiment le niveau de sécurité réel des logiciels mais plutôt le niveau de réactivité de la communauté/des entreprises, à sortir des patchs sur leur logiciel. Cette analyse montre donc bien l'impact que peut avoir la communauté sur la sécurité des logiciels. Et cela peut donc expliquer pourquoi, dans notre analyse, nous avons des logiciels open source plus sécurisés car la communauté est plus active pour corriger les bugs.
- [Schryen, 2011], qui réalise une analyse sur les vulnérabilités qui restent non patchées en

fonction de leur sévérité. Cela concorde aussi avec ce que nous disions ci-dessus : une communauté plus active sur les logiciels open source, qui réagit plus vite pour corriger les bugs peut entraîner le fait qu'il y ait moins de bugs sur les logiciels. Il reste donc aussi moins de failles dans ceux-ci.

– [Hoepman & Jacobs, 2013], qui partent d'une réflexion générale sur la sécurité et de différentes définitions et méthodologies utilisées dans l'open source et le closed source.

13. Les limites des résultats

Même si, avec les données que nous avons récoltées, il semblerait que les logiciels open source soient plus sécurisés que les logiciels closed source, il serait erroné d'en faire une généralité. C'est ce que nous allons démontrer dans cette partie.

13.1. Les logiciels open source et closed source

Comme nous avons déjà pu aborder le sujet, nous avons comparé 6 logiciels de 3 branches de l'informatique. Il existe bien sûr d'autres branches et des milliers d'autres logiciels à analyser.

Comme le montre [Schryen, 2011], chaque logiciel est dépendant de son gestionnaire, de son investissement sur le logiciel, de sa philosophie de développement, de la structure du développement, des investissements réalisés[54], etc. Il montre aussi qu'il peut même y avoir des différences entre des logiciels d'un même gestionnaire. Cela lui permet de démontrer que la sécurité d'un logiciel soit indépendante du fait qu'il soit open source ou closed source.

Ce qui veut dire que, pour notre analyse, il se pourrait très bien qu'en choisissant d'autres logiciels, gérés par d'autres personnes, nous aurions des résultats totalement différents. Pour minimiser cette probabilité, il faudrait dès lors pouvoir faire une analyse sur la plupart, et au mieux, tous les logiciels existants. Mais ceci est, comme nous l'avons déjà dit, presque impossible.

13.2. La sécurité d'un logiciel est un ensemble de composants

Comme nous l'avons déjà énoncé, [Schryen & Kadura, 2009] disent que la sécurité est vue comme un seul atome qui, en réalité, doit être découpé en composants qu'il faut analyser indépendamment pour pouvoir déterminer si un logiciel est sécurisé ou non.

Dans ce mémoire, nous avons récolté des données sur deux éléments qui permettent de définir la sécurité d'un logiciel : les failles et les exploits. Comme nous avons déjà pu le voir, d'autres composants ont été abordés dans les différents documents scientifiques que nous avons pu citer. Nous allons d'ailleurs détailler, dans les prochains points, certains qui ont les mêmes résultats que nous, c'est-à-dire que les logiciels open source sont plus sécurisés que les logiciels closed source.

Ce qui fait que, pour réaliser une analyse complète concernant la sécurité, [Schryen & Kadura, 2009] nous informent qu'il faudrait analyser chaque composant ensemble et que l'analyse d'une seule ou d'une petite partie de ces composants ne peut suffire à montrer un résultat correct.

13.3. Le processus de développement.

Notre analyse se base sur l'accessibilité du code source ou non, mais ne porte pas sur le processus de développement d'un logiciel, ou encore sur l'architecture, le design du logiciel, les méthodes utilisées pour le développer, etc. Et tout cela peut varier, d'un logiciel à l'autre. [Schryen, 2011] nous avertit que cela peut même varier entre des logiciels d'un même gestionnaire et ce, en

54 Investissements aussi bien en temps qu'en argent.

fonction de l'expérience qu'il a pu prendre, des objectifs qu'il a pour le logiciel, de l'investissement qu'il met dessus, etc.

Et, comme le rapportent, [Schryen & Kadura, 2009], ces différents éléments, non exhaustifs, influencent aussi la sécurité d'un logiciel.

13.4. La licence utilisée

[Schryen & Kadura, 2009] nous informent que les logiciels open source et closed source dépendent aussi des licences qui leur sont attribuées. Ceci est d'autant plus vrai pour l'open source avec la multitude de licences disponibles[55] qui offrent différents niveaux de contrôle et différentes autorisations. Par exemple, comme nous en informent [Hoepman & Jacobs, 2013], certaines licences obligent de redistribuer le code après modification alors que d'autres refusent la redistribution du code.

C'est pour cela que [Schryen & Kadura, 2009] affirment que pour pouvoir identifier l'impact des différents modèles de logiciels sur la sécurité, il faut être capable de distinguer plusieurs catégories de logiciels en fonction, entre autres, des différences entre les licences, ou même entre les processus de développement, etc. Ces différentes catégories devraient alors être analysées et traitées différemment.

13.5. La publication des failles

Pour nos analyses, nous nous sommes basés sur la liste des failles publiées. Ce qui signifie que l'on est dépendant de la publication de ces failles qui sont disponibles sur [CVE details]. En effet, [Schryen, 2011] nous montre que la sécurité est un point important pour le succès du logiciel sur le marché, qui peut faire en sorte qu'il soit acheté ou non, qui peut soit mener à son succès, soit mener à sa perte.

Selon [Hoepman & Jacobs, 2013], les entreprises, peuvent perdre le respect auprès des utilisateurs et leur crédibilité, surtout si des grosses failles sont découvertes sur des logiciels en closed source. Cela s'explique par le fait que personne ne peut avoir accès au code et donc, qu'on ne peut faire confiance qu'aux développeurs du logiciel. Si on apprend que le logiciel est non sécurisé et qu'il faut attendre que le fournisseur réagisse car personne d'autre ne peut y apporter de correction, le logiciel risque de perdre de la popularité (et encore plus s'il est payant). C'est pourquoi certaines entreprises communiquent très rarement sur les failles trouvées/corrigées.

En ce qui concerne les logiciels open source, [Saffiotti & Awyzio & Brown, 2004] écrivent que la communauté a développé toute une structure et une démarche à suivre pour développer le logiciel. Du point de vue de la publication des failles, cela veut dire, selon [Schryen & Kadura, 2009], qu'il y a de fortes chances pour que des failles soient trouvées, discutées et résolues avant même d'être listées dans la liste des vulnérabilités [CVE].

55 [Schryen, 2011] nous informe que la plupart des licences sont définies par OSI et maintenues par FSF : la « Free Software Foundation »

[Hoepman & Jacobs, 2013] avertissent que certaines failles découvertes et listées ne sont pas toutes nécessairement corrigées. [Schryen, 2011] appelle cela « l'effort économique » : lorsque le gain entre la résolution de la faille est moindre que le gain de sécurité du logiciel, la faille n'est pas corrigée. Autrement dit, si le poids de la faille est faible mais que l'effort et le temps à investir sont énormes, il y a des chances pour qu'on laisse la faille telle qu'elle est. [Schryen, 2011] affirme que c'est le parti pris par [Microsoft] pour développer ses logiciels. [Hoepman & Jacobs, 2013] confirment alors que ces failles ne sont alors pas connues. Ce qui pourrait dès lors expliquer pourquoi il y a tant de failles de poids forts par rapport aux failles de poids faibles concernant les logiciels closed source étudiés qui sont des logiciels [Microsoft].

[Alhazmi & Malaiya & Ray, 2005] nous apprennent aussi que le plus important n'est pas les vulnérabilités publiées car cela signifie qu'elles sont connues. Selon eux, les problèmes et les risques que peut causer un logiciel viennent de l'inconnue et donc, des failles qui ne sont pas encore découvertes. On peut donc en effet connaître et corriger 10 failles de faibles poids, s'il en reste 20 inconnues avec un poids fort, cela reste problématique.

Tous ces différents points montrent donc que le nombre de failles publiées ne représente pas le nombre de failles réelles qui se trouvent sur les logiciels.

D'ailleurs, ceci est démontré par les résultats de l'analyse statistique[56] de [Walden & Doyle & Welch & Whelan, 2009]. Ceux-ci ne correspondent pas aux vulnérabilités qui sont reportées dans la base de données qu'ils ont utilisée : [NVD].

13.6. Les exploits réalisés

Pour réaliser nos analyses, nous nous sommes aussi basés sur les exploits réalisés sur les différents logiciels qui sont disponibles sur [Exploit DB]. Maintenant, comme le montrent [Schryen & Kadura, 2009], [Schryen, 2011] ou encore [Floyd & Harrington & Hivale, 2007], on peut distinguer deux grands types de hackers :

- Les hackers « blancs » : ils n'ont aucune mauvaise intention. Ils hackent pour le plaisir, pour apprendre, pour trouver des failles mais sans but de nuire à autrui, etc. Ils peuvent communiquer les failles aux personnes nécessaires pour qu'ils puissent les corriger.

- Les « mauvais » hackers : ils ont des motivations plus prohibées et souvent à des fins criminelles. Ils hackent pour le gain d'argent, causent des dommages à des systèmes ou à des personnes, etc. Ils vont probablement garder pour eux les failles qu'ils ont découvertes ou ils vont les communiquer à d'autres mauvais hackers tout cela dans le but de lancer une attaque.

Par rapport à ces éléments, on peut donc déduire que les exploits qui se trouvent sur le site d'[Exploit DB] sont ceux des hackers « blancs » car les mauvais hackers n'ont aucune raison de

56 Analyse faite sur des web application en PHP

divulguer leurs exploits sur ce site.

On ne peut donc pas dire que les exploits se trouvant sur [Exploit DB] sont exhaustifs, loin de là.

Toujours en ce qui concerne les exploits divulgués, et en reprenant de nouveau la structure mise en place par les logiciels open source et montrée par [Saffiotti & Awyzio & Brown, 2004], tout comme pour les failles, on peut supposer que les hackers « blancs » trouvant des failles utilisent cette structure pour divulguer leurs exploits et les failles présentes sur le logiciel plutôt que d'aller les divulguer sur une source externe.

Un dernier point concernant les exploits est que, comme le dit [Seebruck, 2015], il existe de multiples motivations. Dans l'analyse des résultats, nous avons entre autres parlé du prestige mais il en existe bien d'autres, comme l'idéologie, le plaisir, le profit, la revanche, etc. Et chaque motivation peut donner plus d'envie pour attaquer des logiciels closed source ou open source. Tout comme pour une analyse sur les failles ou sur les composants de la sécurité, il faudrait pouvoir faire une analyse auprès des hackers sur toutes les motivations qu'ils peuvent avoir pour voir s'ils attaquent plus facilement des logiciels en source ouverte ou des logiciels en source fermée. Mais ces données, comme l'écrivent [Floyd & Harrington & Hivale, 2007], sont difficiles à récolter auprès des concernés.

Ces différents points pourraient expliquer pourquoi, selon nos chiffres, nous avons récolté moins d'exploits concernant les logiciels en source ouverte par rapport aux logiciels en source fermée.

14. Guide à la décision de logiciels

14.1. Conclusion concernant l'open source et le closed source.

Pour faire notre analyse, nous nous sommes basés sur 6 logiciels : le noyau Linux, Windows 7, Libre Office, Microsoft Office, Apache HTTP Serveur et IIS Serveur. Ces 6 logiciels sont répartis dans 3 branches distinctes de l'informatique :lLes systèmes d'exploitation (concernant le noyau Linux et Windows 7), les suites bureautiques (concernant Libre Office et Microsoft Office) et les serveurs web (concernant Apache HTTP Serveur et IIS Serveur).

Nous avons porté notre analyse sur certains éléments qui impactent la sécurité des logiciels, comme le poids des failles en fonction de leur implication sur la sécurité d'un logiciel, les exploits réalisés, etc.

Les résultats de notre analyse amènent à penser que les logiciels en source ouverte sont plus sécurisés que les logiciels en source fermée.

Cependant, nous avons démontré qu'il ne fallait pas en déduire une généralité parce qu'il existe beaucoup d'autres logiciels open source et closed source et beaucoup d'autres éléments de sécurité sont à prendre en compte.

La manière de développer le logiciel et toute l'organisation qu'il y a autour du développement de celui-ci peuvent aussi jouer sur la sécurité d'un logiciel.

Par exemple :

- Le patch d'index et le patch time [Schryen & Kadura, 2009]
- Les vulnérabilités non patchées en fonction de leur poids [Schryen, 2011]
- La densité du code [Alhazmi & Malaiya & &Ray, 2005]
- La technologie et le langage utilisés pour le développement du logiciel [Walden & Doyle & Welch & Whelan, 2009].
- etc.

Autrement dit, rien ne permet d'affirmer que nous n'aurions pas des résultats complètement différents en analysant des données et des logiciels différents.

De plus, comme nous l'avons aussi montré dans le chapitre « *6. Recherches de point de vue* », la plupart des arguments pour ou contre l'open ou le closed source ne sont pas mesurables et donc vérifiables via des données concrètes. Ce qui veut dire que, quand bien même le type d'ouverture du code source d'un logiciel aurait une influence sur la sécurité d'un logiciel, il n'est pas possible de le

prouver via des chiffres.

Par exemple :

Deux personnes, ayant le même niveau de programmation, développent toutes les deux un logiciel. La première décide qu'il sera en open source. La seconde, en closed source. En reprenant certains arguments que nous avons énoncés, certains diront que le logiciel open source est souhaitable car il va permettre à d'autres de repérer des failles et donc, de les corriger plus rapidement. D'autres affirmeront que c'est le logiciel closed source qui est préférable car les hackers n'auront pas accès au code source et ne sauront pas voir directement les failles du logiciel. Qui dit vrai ? Comment mesurer ces affirmations ? Lequel de ces deux éléments aura le plus d'impact sur la sécurité du logiciel ? Il n'est pas possible de le savoir.

14.2. Le but du guide à la décision

En partant du principe qu'il y a une multitude d'informations à prendre en compte pour déterminer la sécurité d'un logiciel, et qu'il est même presque impossible de tous les prendre en compte lors d'une comparaison, nous avons décidé de réaliser un guide à la décision. Son but sera de pouvoir comparer deux logiciels quelconques selon différents critères de sécurité et de définir, selon les critères choisis, quel logiciel peut être considéré comme étant le plus sécurisé.

Pour cela, nous allons tout d'abord définir un tableau de mesures. Dans ce tableau se trouveront différents arguments de sécurité qui ont été choisis. Ces arguments devront pouvoir être mesurables. Cette mesure, nous l’appellerons, comme pour les différents modèles que nous avons pu rencontrer, « le poids ».

Ce tableau devra être assez flexible et s'adapter en fonction des arguments que l'on veut analyser.

Par exemple :

– 5 mesures ont été définies dans les arguments. Les données de chaque argument sont encodées. Le tableau doit alors tout prendre en compte.

– 5 mesures ont été définies dans les arguments. Seulement, on ne veut comparer que 2 mesures de sécurité, la comparaison avec le reste ne nous intéresse pas. Le tableau ne doit alors prendre en compte que les deux éléments qui sont encodés.

– 5 mesures ont été définies dans les arguments. On voudrait prendre en compte 2 nouvelles mesures prises à partir de nouveaux arguments. En suivant la méthodologie définie, il devrait pourvoir être possible d'ajouter ces nouveaux éléments pour que le tableau puisse en prendre compte.

Ainsi, en définissant, pour un logiciel, chaque mesure sélectionnée, nous obtiendrons son « tableau de mesures ». En pratiquant la même démarche pour un second logiciel, nous obtiendrons

un second tableau de mesure.

En comparant ces deux tableaux obtenus, il sera possible de déterminer, pour chaque mesure, quel est le logiciel le plus sécurisé. Et au final, le résultat de la comparaison de chaque mesure permettra de déterminer lequel des deux logiciels est le plus sécurisé. Retenons également qu'il est possible d'obtenir une égalité.

Par exemple :

- 5 mesures ont été effectuées. Le logiciel 1 a quatre meilleures mesures. Le logiciel 2 n'a donc qu'une meilleure mesure. Le logiciel 1 peut dès lors être considéré comme plus sécurisé.

- 6 mesures ont été réalisées. Le logiciel 1 et 2 ont chacun 3 mesures. Ils sont donc à égalité.

14.2.1. Le guide à la décision vis-à-vis de la comparaison de logiciels open source et closed source

Une fois défini, le guide à la décision nous servira pour fixer plusieurs arguments de sécurité et mesures.

Nous essayerons de nous diversifier dans les différents arguments de sécurité. Nous reprendrons certains que nous avons déjà utilisés dans ce document mais aussi de nouveaux. Ceci dans le but de comparer de nouvelles données et mesures.

Les mesures pourront être, elles aussi, reprises des analyses que nous avons déjà pratiquées, soit nouvelles. Nous allons aussi diversifier nos sources.

Nous allons ensuite comparer nos 6 logiciels choisis par rapport à ces nouveaux éléments définis dans notre guide à la décision. Nous pourrons ainsi voir, si, en tenant compte de nouveaux arguments, de nouvelles mesures, de nouvelles données, de nouvelles sources, etc, nous obtenons les mêmes résultats que précédemment. Ou, au contraire, obtenir des résultats différents, ce qui montrerait en effet, qu'en fonction des données comparées et sources utilisées, les conclusions ne sont pas les mêmes.

14.3. La méthodologie utilisée

Nous allons, dans cette partie, définir la méthodologie utilisée pour construire le tableau de mesures.

14.3.1. Définition des catégories

[Jonsson & Stömberg & Lindskog, 1999] nous montrent qu'un logiciel a toujours un état, c'est-à-dire une phase dans laquelle peut se trouver le système à un moment donné. Ils définissent la provenance de cet état en fonction de deux influences :

- L'état interne : lorsque l'état du système provient du système lui-même

Par exemple :

- Codes.
- Variables.
- Fonctions.

- L'état externe : lorsque l'état du système provient d'une influence externe au système.

  Par exemple :

  - Action humaine.
  - Action d'un autre logiciel.

C'est par rapport à cela que nous allons nous baser pour définir nos catégories qui contiendront nos arguments de sécurité. Ces catégories seront dès lors au nombre de deux et définiront d'où provient l'influence de la sécurité du logiciel :

- **La sécurité interne :** Comprend les éléments internes au logiciel.

  Par exemple :

  - Les failles de celui-ci.

- **La sécurité externe :** Comprend les éléments externes au logiciel.

  Par exemple :

  - Le fait qu'un logiciel soit toujours mis à jour.

14.3.2. Définition des arguments de sécurité

Pour réaliser le guide à la décision, il faut commencer par définir différents éléments que nous allons appeler « Arguments de sécurité ». Et chacun de ces arguments sera classé dans une des deux catégories que nous avons définies ci-dessus.

Par exemple :

- Les failles du logiciel.

Pour notre comparaison, nous les choisirons par rapport à ce que nous avons pu lire. Nous les avons déjà abordés, pour certains, dans ce document.

14.3.3. Attribution d'une mesure aux arguments

Comme nous l'avons déjà évoqué, pour pouvoir prouver quelque chose et, en ce qui nous concerne, pour pouvoir prouver la sécurité d'un logiciel, il faut qu'il soit possible de la mesurer. C'est l'élément principal qui nous a permis de savoir si un argument était valable ou non : est-ce qu'il est mesurable ? Si c'était le cas, on peut reprendre l'argument. Si ce n'est pas le cas, il ne faut pas le prendre en considération.

La liste des arguments que nous allons définir dans les prochains chapitres répondra aux critères ci-dessous et nous débattrons sur la mesurabilité de chacun d'eux.

Si les arguments sont mesurables, il faut dès lors définir les données à mesurer.

Par exemple :

- L'année de début et de fin de prise en compte des données.
- Les vulnérabilités trouvées.

À côté de la définition de la mesure, nous allons ajouter deux éléments qui peuvent caractériser l'état de sécurité du logiciel.

Le premier élément sera « *non définissable* ». Nous prenons en compte cet élément suite à une remarque que nous avons déjà faite dans les limites des résultats et qui est « *la disponibilité de l'information* ». C'est aussi un élément qui a été pris en compte par [Schryen, 2011] dans son analyse : le fait qu'il ne soit pas possible de récolter une information pour répondre à la demande. Il a même rejeté des logiciels dans sa comparaison car il n'avait pas assez d'informations disponibles. Nous procéderons de la même façon ici : s'il n'est pas possible, pour un logiciel ou un autre, de trouver la réponse à un argument, ce dernier sera classé comme étant « non définissable ». Il ne sera donc pas pris en compte dans le résultat final pour le logiciel en question.

Par exemple :

- L'argument est « le nombre d'exploits portés sur le logiciel ». Il n'est pas possible de trouver une réponse. On marque donc l'argument comme étant « indéfinissable ».

Cet élément peut aussi être utilisé si l'on ne veut pas comparer l'argument pour l'analyse que l'on veut réaliser[57].

Par exemple :

- 10 arguments ont été définis mais on ne veut en comparer que deux. Les huit restants peuvent être marqués comme « non définissable », ils ne seront dès lors pas pris en compte dans la comparaison finale.

Le deuxième élément sera « *non sécurisé* ». Nous prenons cet élément en compte suite aux différentes phases de développements énoncées par [Alhazmi & Malaiya & Ray, 2005] que nous avons déjà évoquées et sur lesquelles nous reviendrons plus tard dans ce document. Via ces phases, nous nous sommes posé la question suivante : Que se passe-t-il lorsque le logiciel est arrivé à la fin de la dernière phase, qu'il est en fin de vie et n'est plus du tout maintenu par les développeurs et donc, n'est plus du tout à jour. C'est pour ce genre de cas que nous avons défini l'élément « *non sécurisé* » : si, pour une raison ou une autre, un seul argument est défini comme « non sécurisé », et quel que soit le résultat des autres arguments, le logiciel sera défini comme ne devant pas être utilisé

57 Cela ne sera pas notre cas lors de la comparaison de nos logiciels

car on ne peut pas en garantir la sécurité.

Par exemple :

– L'argument est « la phase de développement du logiciel » et le logiciel analysé est Windows 95. Il est de notoriété publique, et [Alhazmi & Malaiya & Ray, 2005] le confirment, que ce système d'exploitation n'est plus en cours de développement. Il n'est plus maintenu, ses failles ne sont donc plus corrigées. L'argument est marqué comme « non sécurisé » et cela voudra donc dire que le logiciel ne doit absolument plus être utilisé.

En résumé, pour un argument de sécurité défini, on peut choisir entre 3 éléments. Un seul élément peut être sélectionné à chaque fois. Ces éléments sont :

– Pouvoir mesurer l'argument et encoder les données récoltées.

– Définir l'argument comme étant « non définissable ».

– Définir l'argument comme étant « non sécurisé ».

14.3.4. Attribution d'un poids aux mesures

Avec les mesures récoltées et définissables, il est possible de déterminer un certain poids. Ce poids sera utilisé pour la comparaison des données avec un autre logiciel.

Par exemple :

– Un logiciel avec 10 failles de poids 2 aura un nombre de 20. Mais un logiciel avec 20 failles de poids 10 aura un nombre de 200.

– Un logiciel toujours maintenu par ses développeurs qui font des mises à jour régulièrement aura un nombre de 2. Mais un logiciel en fin de vie, où les développeurs sont passés à autre chose et sur lequel ils ne font plus que des mises à jour occasionnelles aura un nombre de 3.

14.3.5. Déterminer la façon d'attribuer les données

Les arguments qui seront choisis seront tous différents les uns des autres. Les éléments caractérisant leur sécurité seront aussi, pour la plupart, différents.

Par exemple :

– Pour définir la mesure de l'argument « Le poids des failles », nous utilisons l'indice [CVSS], qui est calculé de 1 à 10

– Pour définir la mesure de l'argument « le taux des exploits », nous allons prendre toutes les attaques réalisées sur un certain laps d'années, que nous allons diviser par le nombre d'années en question, pour avoir la moyenne d'exploits réalisés sur un an.

Il n'est donc pas possible de déterminer une seule et même manière de définir comment attribuer une mesure pour ces arguments. Tout comme, par le même fait, il n'est pas possible de déterminer une seule et même manière de définir comment l'argument peut être défini comme « non

sécurisé ».

C'est pourquoi il faut définir, pour chaque argument, la manière d'effectuer la mesure ou la manière de le définir comme « non sécurisé ». Il faut aussi définir comment la mesure est calculée pour obtenir le poids final de l'argument.

Le but ici est que, tout un chacun, connaissant le domaine de l'informatique ou non, puisse compléter correctement les données. C'est pourquoi il faut donner un maximum d'informations possibles pour trouver les données d'un argument pour les logiciels étudiés. Maintenant, il est aussi possible que, pour certains arguments, l'utilisateur voulant réaliser une comparaison de logiciel doive effectuer des recherches par lui-même pour accéder à l'information. Mais tout cela doit être expliqué lors de la description des arguments.

En ce qui concerne la définition du point « non sécurisé » pour un logiciel, il est possible qu'un argument ne remplisse jamais cette condition. C'est-à-dire que l'argument sera soit défini par une valeur, soit attribué comme étant « non définissable ». Mais il ne sera jamais « non sécurisé ».

Par exemple :

– En ce qui concerne l'argument « le taux des exploits », il est possible de déterminer une manière de calculer la mesure de l'argument. Par contre, il n'est pas possible de définir à partir de quel moment l'argument peut être marqué comme « non sécurisé ». En effet, il n'est pas possible d'établir un nombre ou une donnée récoltable qui permette de dire à partir de quel seuil le taux des exploits réalisés sur le logiciel fait que celui-ci ne doit plus être utilisé du tout. Cet argument ne peut donc être qu'une donnée comparable.

La seule chose qui sera en commun pour chaque argument de sécurité est sa définition comme étant « non définissable ». Cet élément sera choisi, pour chaque argument, s'il n'est pas possible d'en définir une mesure ou de le définir comme étant « non sécurisé ».

14.4. Encoder les données récoltées

Pour encoder les différentes données récoltées pour les différents arguments d'un logiciel, nous avons établi un document tableur intitulé « Arguments_Securite_Logiciel-Template.ods »[58]. Il suffit donc d'en faire une copie pour y encoder les données du logiciel retenu. C'est donc un document par logiciel. Il faut donc effectuer autant de copies du tableur que de nombre de logiciels que l'on veut analyser.

14.4.1. Fonctionnement du fichier

Le fonctionnement du fichier est assez simple : vous y trouverez plusieurs feuilles de tableur.

La première feuille s'intitule « Récapitulatif » et reprend les données qui sont encodées dans les

58 Ce document a été réalisé avec Libre Office, version 4.2.8.2. Nous recommandons d'utiliser ce logiciel avec cette version ou supérieure pour le bon fonctionnement de celui-ci.

autres feuilles. Ces dernières reprennent chacune les données d'un argument de sécurité. Il y a donc une feuille par argument de sécurité défini.

14.4.2. Encodage des données

De manière générale, les feuilles sont à encoder de manière suivante :

– Tous les « *0* » qui sont marqués en italique sont des données qu'il est possible de modifier ou non, selon les informations récoltées. Leur titre est en petite police et en gras.

– Les « *0* » qui sont marqués en italique et soulignés sont des données qu'il ne faut pas modifier. Ces champs seront calculés automatiquement en fonction des données qui seront encodées. Leur titre a une plus grande taille et est en gras et italique.

Par exemple :

– Sur la page « Récapitulatif », le seul champ à encoder est le nom du logiciel étudié. Tous les autres seront modifiés automatiquement en fonction des données encodées dans les autres feuilles.

Logiciel étudié :	*0*

SE – Mesure de sécurité	**Poids obtenu**	**Non définissable**	**Non sécurisé**
Phase de développement du logiciel	*0*	*0*	*0*

– Sur la page « SE – Phase de développement du logiciel », tous les champs sont à renseigner, sauf le poids total, qui lui, est calculé en fonction de ce qui est encodé.

Non définissable	*0*
Non sécurisé	*0*

Phase numéro	**Choix**
1	*0*
2	*0*
3	*0*
Poids total	*0*

Si un élément est défini comme « Non définissable » ou comme « Non sécurisé », il suffit de remplacer le 0 par le 1. Pour le reste des données, il suffit de suivre ce qui est défini sur la note de l'argument et qui est disponible en 3ème ligne de chaque feuille.

Par exemple :

– Sur la page « SE – Phase de développement du logiciel» :

Notes :

Pour la donnée sélectionnable, il suffit de remplacer le 0 par le 1
Pour rappel :
- Phase 1 : Le système commence à être déployé, les utilisateurs commencent à l'utiliser et les premières vulnérabilités commencent à être découvertes.
- Phase 2 : Le système arrive à son pic de popularité, il est de plus en plus utilisé et l'effort apporté par les développeurs concernant la correction des vulnérabilités est à son maximum.
- Phase 3 : Le système commence à être remplacé par un autre. Les utilisateurs commencent à le quitter pour en utiliser un autre, la fréquence des patchs diminue ainsi que le support d'aide du logiciel.

Notons que si tous les champs sont encodés, l'ordre de sélection pour le récapitulatif de la première feuille est le suivant : « Non définissable » ⇒ « Non sécurisé » => « Données ».

Par exemple :

– Si les champs « Non définissable » et « Non sécurisé » sont encodés à « 1 », sur la page du récapitulatif, seul le champ « Non définissable » sera à 1 et le champ « Non sécurisé » sera à 0.

Autrement dit, pour que les données d'un argument soient prises en compte dans le récapitulatif de la première page, il faut que les données « Non définissable » et « Non sécurisée » restent marquées comme 0.

Si un argument ne permet pas d'encoder le champ « Non sécurisé », celui-ci ne sera pas disponible sur la feuille de cet argument.

Par exemple :

– Sur la page « SI – Le poids des failles d'un logiciel » :

SI – Le poids des failles d'un logiciel

Notes :

Il faut tout d'abord encoder l'année de début et l'année de fin (cela peut être la même année).
Pour marquer le poids des failles, il suffit de mettre un 1 dans la colonne correspondant au poids de la faille.
Une ligne correspond à une faille

Non définissable *0*

Année Début : *0*
Année Fin : *0*

Le champ « Non sécurisé » ne s'y trouve pas. Cela veut dire que sur la feuille du récapitulatif, il restera donc toujours à « 0 ».

14.5. Définitions des arguments de la catégorie « sécurité externe »

Nous allons définir dans cette partie les différents arguments de sécurité externe dont nous tiendrons compte pour notre comparaison de logiciels open source et closed source.

14.5.1. La phase de développement du logiciel

Cet argument de sécurité est lié à l'analyse qu'ont pu faire [Alhazmi & Malaiya & Ray, 2005] sur les différentes phases de développement des logiciels [Microsoft] et qui peut être étendue à d'autres logiciels.

Comme ils ont pu le démontrer dans leur analyse, les phases de développement ont un impact sur les vulnérabilités et leur découverte. Cela serait contrôlé par deux facteurs :

- Le moment où le marché découvre le logiciel et commence à l'utiliser.
- La saturation due à un nombre fini de vulnérabilités.

Les phases de développement ont aussi un impact sur l'attention apportée par les développeurs sur le logiciel pour détecter et corriger les vulnérabilités. En effet, un logiciel en fin de vie recevrait moins de patchs et aurait un support moins important dû au fait que les ressources humaines et/ou techniques sont déployées pour un autre logiciel. Un logiciel en début de vie, au contraire, recevrait de plus en plus de ressources au fur et à mesure de son avancement.

Définition des mesures et des poids

Pour définir la mesure, nous allons reprendre les différentes phases définies par [Alhazmi & Malaiya & Ray, 2005] et qui sont :

- Phase 1 : Le système commence à être déployé. Les utilisateurs commencent à l'utiliser et les premières vulnérabilités commencent à être découvertes.
- Phase 2 : Le système arrive à son pic de popularité, il est de plus en plus utilisé et l'effort apporté par les développeurs concernant la correction des vulnérabilités est à son maximum.
- Phase 3 : Le système commence à être remplacé par un autre. Les utilisateurs commencent à le quitter pour en utiliser un autre, la fréquence des patchs diminue ainsi que le support d'aide du logiciel.

Nous allons donc définir 3 poids : 1, 2 et 3. Ces poids correspondront chacun à une phase de développement qui seront attribués de la manière suivante :

- Le poids 1 sera attribué à la phase 2. Cette phase est la phase où le logiciel est le plus utilisé et où les vulnérabilités peuvent être trouvées plus rapidement mais elle est surtout celle où les développeurs sont les plus réactifs et donc, où les failles peuvent être corrigées plus rapidement.
- Le poids 2 est attribué à la phase 1. Cette phase a un poids plus élevé que la phase 2 tout simplement parce que le logiciel n'a pas encore atteint sa capacité maximale d'utilisation et l'attention maximale de la part des développeurs. De plus, il est toujours possible que, pour diverses raisons comme, par exemple, le manque de succès du logiciel, le développement d'un logiciel ne passe pas cette phase.
- Le poids 3 est attribué à la phase 3. Cette phase a le poids le plus lourd étant donné que

le logiciel est voué à ne plus être maintenu. Il vaut donc mieux commencer à lui trouver un remplaçant.

Définition du « non sécurisé »

Comme nous l'avons déjà abordé lors des exemples, la définition de cet argument comme étant « non sécurisé » tient d'une phase qui n'est pas prise en compte par [Alhazmi & Malaiya & Ray, 2005]. Cette phase est le fait que le logiciel est à l'abandon, plus personne ne s'en occupe et plus aucune mise à jour n'est mise en place pour les éventuelles failles découvertes.

De plus, comme le citent [Alhazmi & Malaiya & Ray, 2005], il faut prendre en compte le fait qu'il peut toujours exister des failles qui ne sont pas encore découvertes. Autrement dit, un logiciel ne peut jamais être considéré comme fiable à 100 %. Ce n'est donc pas parce qu'il n'y a plus aucune faille divulguée sur un logiciel qu'on peut l'utiliser, même si plus aucune mise à jour n'est faite dessus car il est toujours possible d'en découvrir par après.

Cela montre bien que lorsqu'un logiciel n'est plus maintenu, la meilleure chose qu'il reste à faire pour éviter tous problèmes du point de vue de la sécurité est de ne plus l'utiliser.

Où trouver l’information ?

Il n'y a pas de manière définie et unique qui permette d'accéder à cette information pour tous les logiciels. Pour savoir où en est chaque logiciel dans sa phase de développement, il faut donc procéder à quelques recherches. Voici quelques conseils qui permettront d'accéder à l'information :

– Tout d'abord, regarder au niveau des notes de version du logiciel, pour autant qu'il en existe une. Une simple visite sur un moteur de recherche quelconque permet normalement d'y accéder. De là, il est possible de regarder si le logiciel est bien toujours maintenu, quelle est sa version actuelle, etc. Cela permet aussi de savoir si le logiciel que l'on utilise est bien la dernière version disponible. Et donc, que toutes les mises à jour de sécurité ont bien été installées.

Cependant, il faut savoir que d'anciennes versions d'un logiciel peuvent continuer à être mises à jour. Autrement dit, ce n'est pas parce qu'un logiciel est en version 5.X.X que la version 4.X.X n'est plus maintenue. [Alhazmi & Malaiya & Ray, 2005] disent même que, bien souvent, dans le développement d'un logiciel, l'attention portée par les développeurs à une version N est aussi portée à la version N-1. Ils expliquent cela par le fait que les utilisateurs ne font pas directement les mises à jour mais aussi parce qu'il y a un phénomène de reprise de code : il existe du code en commun entre différentes versions et il est possible qu'une faille d'une ancienne version se retrouve aussi dans la nouvelle version. D'où l'intérêt de rester au courant de ce qu'il se passe dans les anciennes versions.

– Il est aussi possible de voir où en est le développement d'un logiciel sur le blog des développeurs et/ou sur le forum du logiciel. Bien souvent, il est possible d'y récolter des informations sur les versions du logiciel actuel, sur les versions en développement, encore maintenues ou dépassées.

Notes sur les versions d'un logiciel.

Nous allons terminer ce point en formulant une remarque concernant les versions d'un logiciel. En effet, il faut pouvoir distinguer ce que l'on appelle un nouveau logiciel d'une nouvelle version d'un logiciel.

Nous allons prendre l'exemple des systèmes d'exploitation de [Microsoft] : Windows. Cet exemple est cité par [Alhazmi & Malaiya & Ray, 2005]. Windows Vista et Windows 7 sont considérés comme deux systèmes d'exploitation différents. Cependant, du code de Windows Vista a été utilisé pour développer Windows 7. Windows 7 pourrait alors être considéré comme simplement une nouvelle version de Windows Vista. On les considère comme deux systèmes d'exploitation tout simplement parce qu'ils vont suivre leur propre cycle de vie et leur propre phase de développement.

De même pour les nouvelles versions d'un logiciel. Bien souvent, lorsqu'un logiciel passe d'une version 4.X.X à une version 5.X.X, il va suivre un nouveau cycle de vie et pourrait être considéré comme un nouveau logiciel. Pour les logiciels gratuits, il est simplement considéré comme une nouvelle version, tout simplement parce qu'il est possible de recevoir la mise à jour directement et gratuitement. Pour savoir cela, de nouveau, il va falloir chercher l'information.

Le but de cette partie étant tout simplement de trouver où en est la phase de développement du logiciel étudié. Si un doute subsiste, rien n'empêche de marquer cet argument comme étant « non définissable ».

14.5.2. Le pourcentage de patchs réalisés

Comme le montrent [Schryen & Kadura, 2009], [Schryen, 2011], [Hoepman & Jacobs, 2013] ou encore [Alhazmi & Malaiya & Ray, 2005], les patchs apportent des corrections et des améliorations au niveau des logiciels. Ils ne reflètent pas, à proprement parler, le niveau de sécurité d'un logiciel mais ils peuvent influencer celui-ci. En effet, il est possible qu'un patch corrige des bugs mais aussi, malencontreusement, en crée de nouveaux.

Cela montre bien l'importance que les patchs peuvent avoir dans la sécurité d'un logiciel et donc, l'importance de prendre en compte cette information dans un tableau de comparaison.

Avoir mis cet argument dans la catégorie des processus de développement vient du fait que, comme nous en informe, entre autres, [Seebruck, 2015], la gestion des failles et des attaques engendre un coût non négligeable. Et bien souvent, lorsqu'il y a des risques au niveau d'un logiciel, le but d'un gestionnaire n'est pas d’éliminer le risque entièrement mais bien de réduire le risque dans un coût qui peut être rentable.

Cela est appuyé par [Schryen, 2011] qui montre que même si la plupart du temps, lorsqu'une vulnérabilité est publiée, le gestionnaire souhaitera la corriger le plus rapidement possible. Mais il est possible que, pour des raisons économiques, le gérant du logiciel décide de ne pas la corriger. Cela se déroule dans les cas où la vulnérabilité n'est pas critique, comparé à sa correction qui

engendrait beaucoup de temps et d'argent.

En conclusion, comme le disent [Schryen & Kadura, 2009], les patchs ne reflètent pas le niveau de sécurité d'un logiciel mais bien le niveau d'activité de la communauté à sortir des patchs.

Définition des mesures et des poids

[Schryen & Kadura, 2009] nous montrent qu'il peut y avoir plusieurs données à analyser via les patchs :

- Le nombre de patchs effectués.
- La fréquence des patchs.
- Le temps de réaction entre la détection d'une vulnérabilité et son élimination.
- Le taux des vulnérabilités corrigées en fonction de leur poids.
- Le nombre de failles restantes.
- etc

Pour notre comparaison, nous n'allons pas définir chacun de ces éléments. Cela prendrait trop de temps à réaliser mais aussi beaucoup de temps à trouver les informations nécessaires pour compléter les données. C'est pourquoi nous allons nous concentrer sur les deux mesures suivantes :

- Le nombre de vulnérabilités détectées sur le logiciel.
- Le nombre de patchs effectués. Autrement dit, le nombre de vulnérabilités corrigées.

Grâce à ces deux mesures, nous pourrons connaître le pourcentage de patchs effectués par rapport au nombre de vulnérabilités détectées.

C'est donc ce qui constituera le poids de notre argument. Le logiciel avec le pourcentage le moins élevé sera considéré comme étant plus sécurisé pour cet argument.

Par exemple :

- Un logiciel qui a 856 failles détectées avec 125 patchs réalisés. Son pourcentage de correction effectuée est donc de 14,60 %.

Définition du « non sécurisé »

Nous n'avons pas trouvé de mesures qui permettraient de définir cet argument comme étant « non sécurisé ». En effet, ce n'est pas parce qu'un logiciel n'a pas de patchs actuellement qu'il n'en aura pas dans le futur.

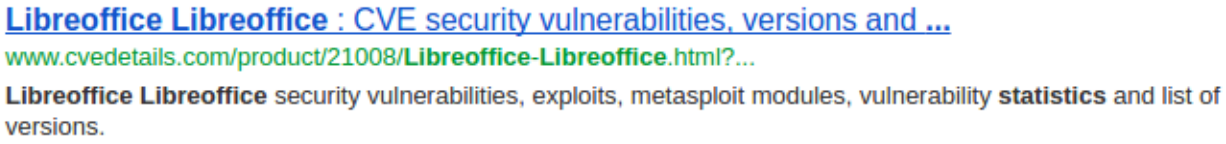
De plus, si un logiciel n'a jamais de patch, c'est qu'il n'est plus maintenu. Si c'est le cas, ce sera le point précédent « la phase de développement du logiciel » qui sera marqué comme étant « non

sécurisé ».

Où trouver l'information ?

Pour définir les deux mesures que nous avons sélectionnées, il est possible de trouver les données via le site [CVE details]. Pour cela, il suffit d'effectuer une recherche sur les statistiques du logiciel que l'on souhaite dans la barre des recherches qui se situe en haut à droite du site :

Et de cliquer sur le lien s'intitulant « |Le nom du logiciel recherché| : CVE security vulnerabilities, version and... ».



On arrive sur la page des données du logiciel en question.

Les données que nous allons utiliser sont celles qui se trouvent au-dessus de la page :

Dans le cas cité ci-dessus, le logiciel a 2 vulnérabilités définies ainsi que 32 patchs.

Ces données sont définies par « *OVAL* »[59] qui est une communauté internationale qui promeut la publication et la disponibilité des contenus de sécurité. Leur but est aussi de standardiser le partage d'informations entre différents outils et services de sécurité. Le langage définit 3 étapes de processus d'évaluation :

- La représentation des informations de configuration pour les tests de systèmes.
- L'analyse du système en fonction de son état actuel (analyse des vulnérabilités, de la configuration, des patchs effectués, etc).
- Le report des résultats de l'évaluation.

Toutes les définitions déterminées par cette communauté, dont la source provient de différents sites que nous avons pour certains déjà évoqués[60], sont recensées par le site [ITSecDB]. C'est d'ailleurs sur ce site que pointent les liens de [CVE details].

Revenons-en à nos moutons. En ce qui concerne la collecte des données, il faut bien faire attention à prendre les deux données mentionnées ci-dessus et non pas, par exemple, le nombre de vulnérabilités totales que l'on peut trouver sur le tableau du logiciel :

59 Open Vulnerability and Assessment Language
60 http://www.itsecdb.com/oval/oval-sources-index.php

# of Vulnerabilities
2
5
3
5
2
17

En effet, étant donné que les données de la diffusion des patchs sont disponibles, via [CVE details], sur [ITSecDB], il est logique que, pour récolter les données des vulnérabilités, on utilise la même source. Et donc, que pour ces deux données, on utilise les définitions des failles et patchs fournies par « *OVAL* » pour être certain d'avoir les mêmes critères de sélection.

S'il y a plus de vulnérabilités définies dans le tableau récapitulatif dans le tableau de [CVE details], c'est que des vulnérabilités ne sont pas recensées sur [ITSecDB] et/ou ne sont pas définies par « *OVAL* ». Cela montre bien que les critères entre les deux bases de données de vulnérabilités ne sont pas les mêmes.

Pour obtenir une comparaison des plus correctes, mieux vaut donc baser les données récoltées sur une même source qui aura les mêmes critères pour définir les failles et les patchs plutôt que sur deux sources différentes qui auront des critères de sélections différentes qui pourraient tronquer les résultats.

D'ailleurs, [ITSecDB] considère une vulnérabilité comme étant une vulnérabilité et un patch comme étant la résolution d'une faille. On a donc bien un ratio de 1 pour 1. On parle donc bien de la même chose et on évite les problèmes comme par exemple, considérer un patch comme étant la résolution d'un ensemble de vulnérabilités et ce, sans savoir vraiment combien.

Il est aussi possible, comme on peut le remarquer sur l'exemple ci-dessus, d'avoir plus de patchs que de vulnérabilités. [Schryen & Kadura, 2009] expliquent cela par le fait qu'il est possible que des failles soient trouvées, discutées et résolues avant qu'elles ne soient ajoutées dans les listes de vulnérabilités. Ce qui peut expliquer le fait qu'un patch soit sorti sans pour autant que le nombre de vulnérabilités soit augmenté.

14.5.3. Les exploits réalisés sur un logiciel

C'est un argument que nous avons utilisé lors de notre analyse. Nous ne reviendrons donc pas sur la nécessité de le prendre en compte dans notre tableau.

Définition des mesures et des poids

En ce qui concerne la mesure, pour ne pas que cela soit trop compliqué et trop long à récolter les données, nous allons la définir sur le nombre d'exploits réalisés. Comparé à ce que nous avons réalisé dans ce document, nous n'allons pas prendre en compte le pourcentage de vérifications faites

par [Offensive Security].

Nous allons aussi prendre en compte le nombre d'années durant lesquelles se sont déroulés ces exploits. Ainsi, cela va permettre de faire une moyenne du nombre d'exploits réalisés sur une année.

Par exemple :

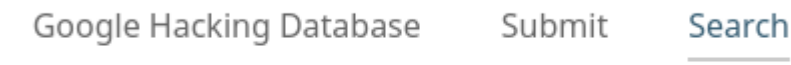
– 525 exploits ont été réalisés entre l'année 2010 et 2016. Cela représente 525 exploits répartis sur 7 années. Cela fait donc une moyenne de 75 exploits par an.

Définition du « non sécurisé »

La valeur du non sécurisé n'est pas d'application pour cet argument. Il n'est en effet pas possible de déterminer une limite du nombre d'exploits qui démontrerait qu'un logiciel soit considéré comme « non sécurisé ».

Où trouver l'information ?

De nouveau, nous allons nous baser sur le site [Exploit DB]. Pour effectuer une recherche, il suffit de cliquer sur le lien « search » qui se trouve en haut à droite du site :

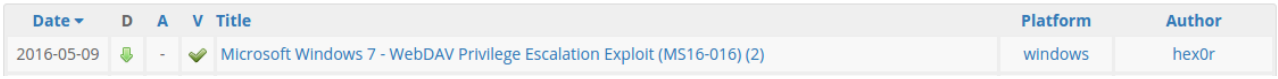


Ensuite, rechercher le logiciel pour lequel on veut réaliser une analyse :

Avec cela, on obtient le tableau des exploits réalisés sur le logiciel en question. On obtient ici déjà deux informations utiles qui sont :

– Le nombre total d'exploits qui ont été recensés. Pour notre recherche, cela donne 98[61].

– L'année de fin de prise en compte des données. Cette année est le premier élément du tableau. C'est donc l'année 2016 dans ce cas-ci.



Pour obtenir l'année de début de prise en compte des données, il suffit d'aller à la fin du tableau et de prendre la dernière ligne. Par exemple, l'année 2000.

61 Recherche effectuée le 14/05/2016. Il est évidemment possible que les données changent au fur et à mesure du temps qui passe.

Par exemple :

- Les données ci-dessous nous donneront comme résultat final le tableau suivant :

Non définissable	*0*
Année Début :	*2000*
Année Fin :	*2016*
Nombre d'années prises en compte	*17*
Nombre d'exploits :	*98*
Moyenne du nombre d'exploits sur un an :	*6*

Autrement dit, une moyenne de 6 exploits par an.

14.6. Définitions des arguments de la catégorie « sécurité interne »

Nous allons maintenant définir les différents arguments de sécurité internes dont nous tiendrons compte pour notre comparaison de logiciels open source et closed source.

14.6.1. Le poids des failles d'un logiciel

De nouveau, c'est un argument que nous avons déjà pris en compte dans notre analyse. Nous n'argumenterons dès lors plus sur son utilité.

Définition des mesures et des poids

Pour définir les différentes données qu'il faut récolter, nous allons établir les mêmes éléments que pour notre analyse. En résumé : il faut prendre comme référence le dictionnaire [CVE] de MITRE. Pour obtenir la sévérité d'une faille, il faut se baser sur l'indice [CVSS] arrondi au plus proche pour avoir un nombre sans virgule entre 1 et 10. Ce nombre fera office de poids pour le calcul de nos données.

En ce qui concerne les mesures, au niveau de notre analyse, nous avons toujours choisi un intervalle fixe d'années pour chaque logiciel. Comme dans ce cas-ci, le tableau devrait pouvoir être utilisé pour n'importe quel type de logiciel (avec beaucoup de données étalées sur beaucoup d'années ou non), nous avons ajouté une donnée qui est l'année de début et l'année de fin de la prise en compte des données. Il faut donc au minimum prendre les failles recensées sur une année complète. Cela permet ainsi de pouvoir réaliser une moyenne de failles sur une année.

Par rapport aux données récoltées dans le tableau de failles, nous allons en tirer 3 éléments pour le tableau récapitulatif :

- « *La moyenne du nombre de failles sur une année* » et ce, sans tenir compte du poids de celle-ci. Pourquoi en tenir compte alors que dans notre analyse, nous ne l'avons pas fait, dû au fait, entre autres, que [Schryen & Kadura, 2009] montraient que la faille n'était pas importante mais qu'il

fallait surtout s'intéresser aux conséquences[62] de celle-ci ? Tout simplement parce que nous avons pu remarquer qu'il y avait une relation entre les failles du logiciel et les attaques qui lui sont portées. Ce qui signifie qu'un logiciel qui a plus de failles se fait plus souvent attaquer. C'est donc par rapport à cela que nous avons décidé de prendre en compte dans notre tableau récapitulatif la moyenne du nombre de failles sur une année.

– « *La moyenne du poids sur une année* ». Nous reprenons donc le poids des failles et donc, la sévérité de celle-ci. En ce qui nous concerne, nous avons pris un même nombre d'années pour les logiciels car nous avions les données sur un même nombre d'années. Mais il est possible que des logiciels soient comparés sur des années différentes.

Par exemple :

- Un logiciel dont on a pu récolter les données sur 2 années comparé à un logiciel dont les données ont été récoltées sur 4 années.

Le fait de réaliser une moyenne du poids des failles sur une année permet donc de mettre les logiciels sur un meilleur pied d'égalité[63].

– « *La moyenne du poids en %* ». Cela reprend ce que nous avons accompli durant l'analyse des failles : reporter le poids des failles sur un nombre de 100 failles. Cela permet, de nouveau, de placer sur un meilleur pied d'égalité la comparaison faite si le nombre de failles récoltées n'est pas le même[64].

Définition du « non sécurisé »

Cet élément n'est pas applicable pour cet argument. En effet, il n'est pas possible d'établir à partir de quel nombre de failles et de quel total de poids les failles rendent le logiciel complètement « non sécurisé ». De plus, les failles ont toujours la possibilité d'être corrigées. Définir un élément comme « non sécurisé » sur quelque chose de temporaire ne nous semble donc pas opportun.

Où trouver l'information ?

Il est possible de récolter les données de différents logiciels sur le site [CVE details].

Pour cela, il suffit d'effectuer, comme déjà expliqué, une recherche au niveau du logiciel que l'on veut étudier. On arrive alors sur le tableau récapitulatif des vulnérabilités du logiciel en question.

62 Et donc au poids

63 Les logiciels ne pourront jamais être sur un même pied d'égalité étant donné que d'autres données doivent être prises en compte comme leur phase de développement que nous avons déjà abordée dans ce document ⇒ Possibilité de récolter plus de failles pour un logiciel en début de vie.
D'où l'intérêt de faire un tableau qui prend en compte plusieurs caractéristiques du développement d'un logiciel.

64 Voir l'argumentation faite à ce niveau au chapitre « *9.1.4. Travailler par rapport à un pourcentage* » de ce document.

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2011	2	1	1	2											
2012	5	4	3	3							1				
2014	3	2	2												
2015	5	4	4	3	3						1				
2016	2	2		2	2										
Total	17	13	10	10	5						2				
% Of All		76.5	58.8	58.8	29.4	0.0	0.0	0.0	0.0	0.0	11.8	0.0	0.0	0.0	

On peut ainsi choisir l'année via la colonne « Year » pour arriver sur le détail des vulnérabilités découvertes pour cette année choisie.

#	CVE ID	CWE ID	# of Exploits	Vulnerability Type(s)	Publish Date	Update Date	Score	Gained Access Level	Access	Complexity	Authentication	Conf.	Integ.	Avail.
1	CVE-2016-0795	119		DoS Overflow Mem. Corr.	2016-02-18	2016-02-25	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete
LibreOffice before 5.0.5 allows remote attackers to cause a denial of service (memory corruption) or possibly have unspecified other impact via a crafted LwpTocSuperLayout record in a LotusWordPro (lwp) document.														
2	CVE-2016-0794	119		DoS Overflow Mem. Corr.	2016-02-18	2016-02-25	9.3	None	Remote	Medium	Not required	Complete	Complete	Complete

Pour chaque faille recensée, il faut prendre la colonne « Score », arrondir le nombre au plus près[65] et noter le résultat dans le tableau prévu à cet effet.

Par exemple :

- Pour la capture d'écran ci-dessus, le résultat du tableur sera :

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1									*1*	
2									*1*	

Pour obtenir des données les plus complètes possible, le mieux est de prendre plusieurs années en compte. Maintenant, le recensement de ces données est assez long et peut paraître assez ennuyant à réaliser. Au final, à chacun de décider le nombre de données qu'il veut récolter pour son analyse de logiciel.

Notons que le tableau que nous avons conçu s'arrête à un nombre de 5 000 failles. Nous pensons que c'est un nombre plus que largement suffisant pour une analyse d'un logiciel[66].

Maintenant, il est toujours possible d'aller au-delà de ce nombre : même si le tableau n'est plus mis en forme, les calculs sont faits pour calculer jusqu'à plus du million de lignes de la feuille du tableur[67].

65 0, 1, 2, 3, 4 sont arrondis à la valeur inférieure
5, 6, 7, 8, 9 sont arrondis à la valeur supérieure

66 À titre d'exemple, nos comparaisons sur 5 années n'ont pas dépassé le nombre de 597 failles.

67 Nous souhaitons bon courage à celle ou celui qui voudra recenser autant de failles pour un logiciel :)

14.6.2. Les lignes de code du logiciel

Nous prenons cet argument suite à la remarque faite par [Schryen, 2011] qui nous informait qu'en théorie, le ratio entre un bug et une ligne de code était de 1/35. Autrement dit, il y aurait un bug toutes les 35 lignes de code. Étant donné que les bugs peuvent être utilisés par des hackers, et donc, devenir des vulnérabilités pour le logiciel, on peut donc déduire que ce nombre de bugs représente en fait le nombre de vulnérabilités potentiel d'un logiciel.

Cela veut donc dire que, plus un logiciel est « gros », plus il est susceptible d'avoir des vulnérabilités. On pourrait donc penser que deux logiciels, qui ne font pas le même nombre de lignes de code, auront un nombre différent de vulnérabilités. Il serait donc « logique » de trouver plus de vulnérabilités sur un « gros » logiciel que sur un « petit ».

Par exemple :

- Nous avons comparé le noyau Linux avec Windows 7 sans tenir compte de leur taille de code. Cependant, on peut aisément penser que, du fait que Windows 7 inclut dans son code une interface graphique qui n'est pas présente de base dans le noyau Linux, il y aura plus de vulnérabilités sur le logiciel [Microsoft].

- [Alhazmi & Malaiya & Ray, 2005] nous informent que le noyau Linux, entre la version 6.2 et la version 7.1 est deux fois plus grand du point de vue de son code sans pour autant avoir augmenté son nombre de vulnérabilités. On peut donc déduire que la version 7.1 est mieux du point de vue de la sécurité car, alors qu'elle a plus de lignes de code et donc, que son potentiel de vulnérabilités aurait dû lui aussi augmenter, celles-ci sont restées en nombre plus ou moins le même.

Cela montre bien que le nombre de lignes de code d'un logiciel est un élément à prendre en compte dans la définition de la sécurité d'un logiciel.

Définition des mesures et des poids

Le lecteur attentif pourra donc facilement savoir que la première mesure que nous allons utiliser pour cet argument est « le nombre de lignes de code » du logiciel. Avec cette donnée, et en prenant en compte du ratio bug/nombre de lignes de code donné par [Schryen, 2011] qui est de 1/35, il est possible de définir le nombre de vulnérabilités potentielles du logiciel.

Par exemple :

- Un logiciel avec 1 000 000 de lignes de code aura un potentiel de 28 571 vulnérabilités.

Maintenant, pour pouvoir réaliser une comparaison correcte entre deux logiciels, il n'est pas possible d'utiliser cette donnée telle quelle. En effet, il ne serait pas correct de dire : plus un logiciel est gros, moins il est fiable. En effet, s'il est plus conséquent, c'est peut-être tout simplement parce qu'il offre des fonctionnalités en plus. Mais il n'en est peut-être pas moins sécurisé pour autant.

C'est pourquoi, pour effectuer une analyse, il serait intéressant de comparer le nombre de lignes

de code et donc, le nombre de vulnérabilités potentielles, par rapport au nombre de vulnérabilités réellement découvertes sur le logiciel. Nous allons donc tenir en compte le nombre total de ces vulnérabilités.

Pour les mêmes raisons que nous avons déjà évoquées précédemment, pour cette donnée, nous allons prendre en compte le nombre d'années qui couvre ces découvertes de failles pour faire une moyenne sur un an.

Par exemple :

– Entre l'année 2010 et 2015, le nombre de vulnérabilités découvertes est de 526. Cela donne une moyenne de 88 vulnérabilités découvertes par an.

Pour terminer, nous allons calculer le pourcentage du nombre de vulnérabilités théoriques par rapport à la moyenne du nombre de vulnérabilités sur une année. Le nombre théorique correspondra à 100 % et la moyenne obtenue au pourcentage.

Par exemple :

– Pour 28 571 vulnérabilités théoriques et 88 vulnérabilités moyennes par an, le pourcentage de vulnérabilités réelles sera de 0,31 %.

C'est cette dernière donnée que nous utiliserons pour faire le poids pour les comparaisons. Le logiciel ayant le plus fort pourcentage sera décrété comme étant moins sécurisé pour cet argument des « lignes de codes du logiciel ».

Définition du « non sécurisé »

Il n'est de nouveau pas possible de définir le fait d'être « non sécurisé » pour cet argument-ci.

Où trouver l'information ?

Il y a donc deux informations à trouver pour cet argument.

Tout d'abord, il faut trouver le nombre de lignes de code d'un logiciel. Pour récolter l'information, il va falloir effectuer une recherche sur internet. L'information sera ± facile à trouver en fonction du type d'ouverture du code source :

– En ce qui concerne les logiciels open source, il sera, normalement, facile de trouver l'information. Si ce n'est pas le cas, il suffit de télécharger le code source du logiciel dans sa version non-minifié[68] et d'ouvrir le projet complet via un éditeur de texte qui permet de connaître le nombre total de lignes de code de tous les fichiers[69]. Pour obtenir des données les plus correctes possible,

68 Un code qui a été minifié est un code qui a été passé par un logiciel pour y enlever tous les éléments non nécessaires comme les espaces, les « enter », les commentaires, etc. En fonction du logiciel utilisé, les variables et fonctions peuvent même être remplacées par de simples caractères : un caractère unique pour chaque variable/fonction. Bien souvent, le code final tient sur une seule ligne.

69 Par exemple : notepad++, geany

des logiciels permettent aussi de supprimer tous les commentaires d'un code[70]. Cela permet de ne pas prendre en compte les commentaires éventuels comme étant une ligne de code réelle.

– En ce qui concerne les logiciels closed source, le fait que le code source ne soit pas disponible risque de rendre l'information plus compliquée à connaître. Bien souvent, il faudra se fier à des données estimées[71]. Comme le disent [Schryen & Kadura, 2009], il est aussi toujours possible d'essayer de décompiler un logiciel code source pour accéder à son code source. Mais il faut savoir que cela risque de vous prendre beaucoup de temps sans pour autant être certain de pouvoir y arriver. Et le but de ce document n'est pas d'inciter le lecteur à réaliser des choses illicites non plus.

Ensuite, il faut trouver le nombre de vulnérabilités découvertes sur un logiciel sur une période donnée. Pour cela, nous allons de nouveau nous baser sur le site [CVE details].

Une fois la recherche effectuée, nous obtenons un tableau des vulnérabilités trouvées :

– Dans la première colonne, il est possible de connaître les années prises en compte.

– Dans la deuxième colonne, sur la ligne des totaux, on peut connaître le nombre de vulnérabilités totales découvertes.

Par exemple :

Year	# of Vulnerabilities
2011	2
2012	5
2014	3
2015	5
2016	2
Total	17

Dans ce cas-ci, l'année de début est 2011, l'année de fin est 2016 et le nombre total de vulnérabilités est de 17.

14.7. Réaliser la comparaison des logiciels

Le but de définir ce tableau de mesures est donc de pouvoir, au final, obtenir un tableau regroupant le niveau de sécurité d'un logiciel. En passant un deuxième logiciel à travers cette liste, on obtient donc un deuxième tableau, qu'il est alors possible de comparer avec le premier.

En reprenant le tableur que nous avons réalisé, nous avons donc deux fichiers contenant les données des deux logiciels utilisés. Pour faciliter la comparaison de ces deux tableurs, nous avons réalisé un nouveau fichier tableur intitulé « Comparaison_Arguments_Securite_Logiciel-Template.ods ».

70 Par exemple : eclipse
71 Qui peuvent être tout aussi difficiles à trouver.

Le but de ce fichier est simple : aller récupérer les données des deux fichiers des logiciels étudiés et comparer, pour chaque mesure, les données récoltées. Cela permet ainsi de déterminer, pour chaque mesure, quel logiciel peut être défini comme étant le plus sécurisé. Le tableur lui accordera donc un point.

Par exemple :

Arguments de sécurité	Logiciel	Poids obtenu	Non définissable	Non sécurisé	Point
Phase de développement du logiciel	*Test 1*	*2*	*0*	*0*	***0***
	Test 2	*2*	*0*	*0*	***0***
Le pourcentage de patchs réalisés	*Test 1*	*3,93%*	*0*	*0*	***1***
	Test 2	*0,56%*	*0*	*0*	***0***

De cette manière, en additionnant pour chaque logiciel les points obtenus lors de chaque mesure, il est possible de comparer les résultats des deux logiciels. C'est ainsi que le logiciel ayant le plus de points sera considéré comme étant le plus sécurisé.

Par exemple :

Totaux

Nom du logiciel	Points obtenus	Non définissable	Non sécurisé
Test 1	*2*	*0*	*0*
Test 2	*3*	*1*	*0*

Résultats

Le logiciel "Test 2" est plus sécurisé que le logiciel "Test 1"

14.7.1. Fonctionnement du fichier

Faire fonctionner le fichier est assez simple. Supposons que nous comparons deux logiciels « Test 1 » et « Test 2 ». On a donc dupliqué le tableur pour récolter les données. Le nom des fichiers est respectivement « Arguments_Securite_Logiciel-Template-Test1.ods » et « Arguments_Securite_Logiciel-Template-Test2.ods ».

Au niveau du fichier de comparaison, 2 champs sont prévus pour indiquer le nom des fichiers qui contiennent les données des logiciels à comparer. Il suffit d'y encoder les chemins relatifs[72] qui permettent d'accéder aux deux fichiers de données. Pour que cela soit plus simple, nous conseillons de mettre les deux fichiers contenant les données du logiciel et le fichier de comparaison dans le même dossier. Ainsi, il suffira d'encoder l'appellation des fichiers **sans l'extension**. De plus, pour éviter tous problèmes, nous conseillons d'éviter de mettre des espaces ou des caractères spéciaux

72 Le chemin est à prendre à partir du tableur de comparaison « Comparaison_Arguments_Securite_Logiciel-Template.ods »

dans le nom des fichiers.

Par exemple :

Logiciel	Chemin
1	*Arguments_Securite_Logiciel-Template-Test1*
2	*Arguments_Securite_Logiciel-Template-Test2*

Et c'est tout pour ce fichier. Il va aller directement rechercher les données qui ont été encodées dans les deux fichiers en question, va automatiquement faire les comparaisons nécessaires, calculer les résultats et les afficher.

Si, après avoir encodé le chemin des fichiers, la valeur « #REF ! » apparaît dans le tableau de comparaison, c'est que le chemin et/ou le nom du fichier ne sont pas corrects. Vérifiez donc le chemin encodé. Vérifiez aussi que les fichiers soient accessibles en lecture et qu'il n'y a pas de problème de droit d'accès.

Si vous changez les données d'un logiciel et que vous voulez actualiser les données du tableur de comparaison, il vous suffit de faire la manipulation suivante : aller dans le menu « Outils » => « Contenu des cellules » => « Recalculer »[73]. En fonction de la configuration du logiciel, vous pouvez aussi appuyer sur la touche « F9 » de votre clavier.

14.7.2. Comparaison des données

Chaque mesure des logiciels est comparée en fonction du poids qui a été obtenu pour la mesure pour attribuer, ou non, un point pour un logiciel. En cas d'égalité dans la mesure, aucun des deux logiciels ne marque de point.

Par exemple :

– Pour « La phase de développement du logiciel », le logiciel qui a le plus petit poids prend le point.

Arguments de sécurité	Logiciel	Poids obtenu	Non définissable	Non sécurisé	Point
Phase de développement du logiciel	*Test 1*	*2*	*0*	*0*	***1***
	Test 2	*3*	*0*	*0*	***0***

– Pour « Le pourcentage de patchs réalisés », le logiciel qui a le plus grand pourcentage de patchs effectués remporte le point.

73 Manipulation à faire à partir de Libre Office

Arguments de sécurité	Logiciel	Poids obtenu	Non définissable	Non sécurisé	Point
Le pourcentage de patchs réalisés	*Test 1*	*3,93%*	*0*	*0*	***1***
	Test 2	*0,56%*	*0*	*0*	***0***

– Lorsque les données sont à égalité, aucun logiciel ne marque de point.

Arguments de sécurité	Logiciel	Poids obtenu	Non définissable	Non sécurisé	Point
Phase de développement du logiciel	*Test 1*	*2*	*0*	*0*	***0***
	Test 2	*2*	*0*	*0*	***0***

Lorsqu'une mesure d'un logiciel est marquée comme étant non définissable, la mesure n'est plus prise en compte. Aucun point ne sera dès lors attribué.

Par exemple :

Arguments de sécurité	Logiciel	Poids obtenu	Non définissable	Non sécurisé	Point
Phase de développement du logiciel	*Test 1*	*2*	*0*	*0*	***0***
	Test 2	*0*	*1*	*0*	***0***

Lorsqu'une mesure est définie comme étant non sécurisée, le point est attribué à l'autre logiciel.

Par exemple :

Arguments de sécurité	Logiciel	Poids obtenu	Non définissable	Non sécurisé	Point
Phase de développement du logiciel	*Test 1*	*2*	*0*	*0*	***1***
	Test 2	*0*	*0*	*1*	***0***

Cependant, lors de la comparaison finale, et ce, comme nous l'avons défini, si un logiciel a une mesure marquée comme non sécurisée, le logiciel sera défini comme non sécurisé. Et ce, quel que soit le résultat des autres mesures.

Par exemple :

Totaux

Nom du logiciel	Points obtenus	Non définissable	Non sécurisé
Test 1	*1*	*0*	*0*
Test 2	*6*	*0*	*1*

Résultats

Le logiciel "Test 2" est défini comme non sécurisé. Le logiciel "Test 1" est dès lors recommandé à l'utilisation

Pour finir, il y a différentes configurations qui pourront faire en sorte qu'il ne sera pas possible de déterminer de logiciel gagnant :

- Lorsque les logiciels ont au final le même nombre de points.

Totaux

Nom du logiciel	Points obtenus	Non définissable	Non sécurisé
Test 1	*3*	*0*	*0*
Test 2	*3*	*0*	*0*

Résultats

Les deux logiciels sont équivalents

- Lorsque toutes les mesures sont marquées comme non définissables.

Totaux

Nom du logiciel	Points obtenus	Non définissable	Non sécurisé
Test 1	*0*	*0*	*0*
Test 2	*0*	*7*	*0*

Résultats

Les deux logiciels ne sont pas comparables car il n'y a aucune donnée à comparer

- Lorsque les deux logiciels sont marqués comme étant non sécurisés.

Totaux

Nom du logiciel	Points obtenus	Non définissable	Non sécurisé
Test 1	*0*	*0*	*1*
Test 2	*0*	*6*	*1*

Résultats

Les deux logiciels sont définis comme « non sécurisés »

14.8. Mise en pratique : comparaison des 6 logiciels étudiés

Maintenant que nous avons défini nos données de logiciels à récolter et notre tableau de comparaison, nous allons le mettre en pratique. Pour cela, nous allons de nouveau comparer les 6

logiciels que nous avons utilisés précédemment dans ce document. Nous allons de nouveau effectuer cette comparaison en fonction des 3 branches de l'informatique que nous avions définies.

14.8.1. Les systèmes d'exploitation

La comparaison porte sur le noyau Linux[74] et sur Windows 7[75].

Les résultats[76] de cette comparaison donnent le tableau suivant :

Totaux

Nom du logiciel	Points obtenus	Non définissable	Non sécurisé
Noyau Linux	*3*	*0*	*0*
Windows 7	*4*	*0*	*0*

Résultats

Le logiciel "Windows 7" est plus sécurisé que le logiciel "Noyau Linux"

Lors de nos précédentes comparaisons, les résultats obtenus étaient assez divergents entre les deux logiciels : Windows 7 avait une moyenne de poids de vulnérabilités ainsi qu'une moyenne de poids des attaques plus grandes que le noyau Linux. Mais ce dernier avait plus d'exploits publiés ainsi qu'une moyenne d'attaques par rapport aux nombres de failles supérieures.

Comme on peut le voir au niveau du tableau des résultats, la comparaison entre les deux logiciels est toujours aussi serrée. Mais Windows 7 obtient au final ce que l'on peut appeler « la victoire » avec 4 points obtenus à 3.

14.8.2. Les suites bureautiques

La comparaison porte sur Libre Office[77] et Microsoft Office[78].

Les résultats[79] de cette comparaison donnent le tableau suivant :

Totaux

Nom du logiciel	Points obtenus	Non définissable	Non sécurisé
Libre Office	*6*	*0*	*0*
Microsoft Office	*0*	*0*	*0*

Résultats

Le logiciel "Libre Office" est plus sécurisé que le logiciel "Microsoft Office"

Pour chacune des comparaisons précédentes, Microsoft Office avait obtenu de moins bons résultats que son concurrent, Libre Office : plus grande moyenne de poids des vulnérabilités, plus grande moyenne de poids des attaques, plus d'exploits publiés, plus grand pourcentage d'exploits par rapport au nombre de failles.

74 Voir Annexe 29 « Analyse des données du noyau Linux »
75 Voir Annexe 30 « Analyse des données de Windows 7 »
76 Voir Annexe 31 « Comparaison noyau Linux et Windows 7 »
77 Voir Annexe 32 « Analyse des données de Libre Office »
78 Voir Annexe 33 « Analyse des données de Microsoft Office »
79 Voir Annexe 34 « Comparaison Libre Office et Microsoft Office »

Ces données se confirment avec notre tableau de comparatif où Libre Office obtient tous les points attribuables.

À savoir qu'il n'y a que 6 points obtenus par rapport aux 7 attribuables dû au fait qu'il y a eu une égalité dans la comparaison de la mesure de sécurité « Phase de développement du logiciel ».

14.8.3. Les serveurs web

La comparaison porte sur Apache HTTP Serveur[80] et IIS Serveur[81].

Les résultats[82] de cette comparaison donnent le tableau suivant :

Totaux

Nom du logiciel	Points obtenus	Non définissable	Non sécurisé
Apache HTTP Serveur	*2*	*0*	*0*
IIS Serveur	*3*	*1*	*0*

Résultats

Le logiciel "IIS Serveur" est plus sécurisé que le logiciel "Apache HTTP Serveur"

Lors de nos précédentes comparaisons, IIS Serveur ne s'en était pas trop bien sorti avec une moyenne de poids de vulnérabilités supérieure à Apache HTTP Serveur, plus d'exploits réalisés avec un poids qui est lui aussi supérieur et un pourcentage d'exploits par rapport au nombre de failles qui est lui aussi, toujours supérieur.

Maintenant, avec les nouvelles données que nous avons définies pour notre tableau de comparaison, on remarque que la tendance s'est inversée et que c'est le logiciel IIS Serveur qui en sort vainqueur avec un point d'avance sur un score de 3 à 2.

Cela montre donc un point que nous avons déjà abordé dans le chapitre « *13. Les limites des résultats* » et cela confirme une remarque émise par [Schryen & Kadura, 2009] ou encore [Alhazmi & Malaiya & Ray, 2005] : le fait de ne pas s'arrêter sur des résultats obtenus pour en faire une généralité mais bien de continuer à vouloir analyser d'autres éléments pour confirmer ou non ces résultats obtenus. Car, en fonction des logiciels et/ou de la mesure que l'on compare, on peut obtenir des résultats différents. Et c'est donc bien le cas ici.

À savoir que pour notre tableau de comparaison, les logiciels se sont retrouvés à égalité sur la mesure « Phase de développement du logiciel ». Nous n'avons pas non plus réussi à trouver l'information du nombre de lignes de code qui composent le logiciel IIS Serveur. L'élément a donc été marqué comme « non définissable » et aucun point n'a été attribué pour cette mesure. Cela explique pourquoi il n'y a que 5 points d'attribués sur les 7 mesures possibles.

14.8.4. Résultats de la comparaison entre logiciels open source et closed source

Nous allons maintenant reprendre nos logiciels étudiés et leur réattribuer leur étiquette d'open

80 Voir Annexe 35 « Analyse des données d'Apache HTTP Serveur »
81 Voir Annexe 36 « Analyse des données d'IIS Serveur »
82 Voir Annexe 37 « Comparaison Apache HTTP Serveur et IIS Serveur »

source et de closed source.

Lorsque l'on a comparé les résultats des logiciels en source ouverte et des logiciels en source fermée lors de nos précédentes comparaisons, nous avions à chaque fois obtenu l'open source en vainqueur. Ce qui nous semblait normal car, pour la comparaison des logiciels, les programmes open source avaient plus souvent de meilleurs résultats que les programmes closed source.

Avec notre tableau de comparaison, Windows 7 et IIS Serveur sont passés devant leur concurrent en source ouverte. Ce sont donc deux logiciels en sources fermées qui sont recommandés sur les trois comparaisons.

Maintenant, en additionnant les points obtenus, nous obtenons le tableau suivant :

Catégorie de logiciel	Points obtenus	Non définissable	Non sécurisé
Open source	*11*		
Closed Source	*7*	*1*	

On remarque que l'open source a plus de points que le closed source avec un résultat de 11 à 7 en sa faveur. Ces résultats s'expliquent simplement parce que les comparaisons où étaient vainqueurs les logiciels en source fermée se jouaient à 1 point d'écart. Tandis que, lorsque le logiciel en source ouverte est sorti gagnant, le nombre de points d'écart était de 6.

De nouveau, cela montre bien ce que nous venons d'énoncer lors du point précédent : en fonction de ce que l'on analyse, des données utilisées, de l'interprétation des résultats, etc, la conclusion peut être complètement différente. Et ce n'est donc pas parce que l'on obtient certains résultats qu'il faut directement en tirer une conclusion générale.

14.9. Comparer les logiciels avec le tableau de comparaison

Nous allons, dans cette partie, entamer la discussion par rapport au tableau de comparaison que nous avons défini, parler de ses limites, de son intérêt, de son évolution, etc.

14.9.1. Limite des mesures de sécurité définies

Pour réaliser notre tableau de comparaison, nous avons donc défini cinq arguments de sécurité qui nous ont permis d'établir sept mesures. Bien évidemment, définir la sécurité d'un logiciel ne se limite pas à cela.

Tout comme la comparaison entre des catégories de logiciels, comme nous avons pu le faire entre les logiciels open source et closed source, la comparaison entre deux logiciels reste un vaste sujet. Pour pouvoir réaliser une comparaison correcte et complète, il faudrait pouvoir recenser tous les arguments de sécurité possibles et inimaginables qui existent.

Beaucoup d'auteurs que nous avons pu lire, comme [Schryen & Kadura, 2009], pensent que c'est même impossible de définir un seul et unique modèle de sécurité qui permette cela. Ils expliquent cela par le fait que, même s'il existe des arguments qui peuvent être spécifiques à tout développement, comme ceux que nous avons pu prendre, chaque branche de l'informatique a ses

spécificités dont il faudrait pouvoir tenir compte dans une comparaison complète. Et il faudrait un temps considérable pour toutes les recenser.

Par exemple :

– [Walden & Doyle & Welch & Whelan, 2009] nous montrent qu'il existe différentes failles qu'il est possible d'exploiter sur un logiciel Web développé en PHP.

Par exemple :

– SQL injections

– Cross site scripting

Ces éléments sont spécifiques au développement de logiciels web et peuvent donc être pris en compte pour comparer la sécurité de deux sites web. Et encore, il est possible qu'un site web soit statique, sans base de données et donc, qu'il ne soit pas possible, par exemple, de faire des injections SQL. De plus, si les deux sites web sont dynamiques et avec une base de données, il est possible que la première DB soit sur un moteur MYSQL et que la deuxième soit sur un moteur PostGreSQL. Il faudrait donc pouvoir prendre en compte aussi les spécificités, failles, et autres de ces différents moteurs.

Par contre, les deux failles possibles précédemment citées ne servent à rien pour faire une comparaison avec des applications bureautiques qui n'a pas d'accès à une base de données.

De plus, plus il y a d'arguments, plus il y aura de mesures. C'est autant de temps en plus qu'il faudra passer pour en rechercher les données des logiciels étudiés.

Pouvoir mesurer la sécurité d'un logiciel via un unique modèle prendrait donc beaucoup de temps, que ce soit dans la conception du dit modèle que dans la recherche des données. Pour le moment, personne n'a pris la peine de réaliser une telle chose.

Cela montre bien que, quel que soit le résultat obtenu d'une comparaison, il faut toujours les nuancer par rapport à tout ce qui n'a pas pu être pris en compte.

14.9.2. Définir les logiciels et arguments à comparer

Comme nous venons de le montrer, il peut exister des milliers d'éléments à prendre en compte lors de la comparaison de logiciel. Et ces arguments varient en fonction des logiciels que l'on compare, mais aussi du but final de la comparaison.

Par exemple :

– Nous allons vouloir comparer la qualité de deux pommes, du point de vue du visuel et du goût. Il faut donc définir des arguments de qualité, comme par exemple, est-ce que la pomme est bien ronde ou encore, est-ce que la pomme est bien mûre ? Par rapport à cela, il est possible de comparer deux pommes.

– Ensuite, nous allons vouloir comparer la qualité de deux poires, toujours selon les

mêmes critères. De nouveau, il faut définir des arguments de qualités, comme par exemple : est-ce qu'elle a une belle forme de poire, est-elle bien mûre ? Il sera ainsi toujours possible de comparer deux poires.

– Nous allons maintenant vouloir comparer la qualité de deux fruits : une pomme et une poire. Dans ce cas, tous les arguments de qualité définis pour la pomme et la poire ne peuvent pas être repris pour réaliser cette comparaison-ci. Ainsi, il n'est pas possible de déterminer la qualité d'une poire par rapport à l'argument de la pomme « est-elle bien ronde ? ».

Donc, il faut donc toujours pouvoir adapter les arguments, les utiliser ou non, en fonction de la comparaison que l'on veut faire.

Et c'est la même chose concernant la comparaison de la sécurité de deux logiciels : il est possible de définir un argument pour une certaine catégorie de logiciel qui ne soit pas possible d'utiliser pour comparer avec un logiciel d'une autre catégorie.

De plus, avec le plugin pour Eclipse « ASIDE[83] » montré par [Xie & Lipford & Chu], on peut remarquer que la sécurité dépend aussi du langage utilisé pour développer un logiciel. En effet, le but de ce plugin est de pouvoir fournir un warning lors de l'analyse du code en temps réel lorsqu'une erreur de programmation, qui peut engendrer une faille dans le logiciel, est détectée. Cependant, cette analyse n'est seulement spécifique que pour le langage JAVA. Ce plugin, étant adapté pour un langage spécifique, n'est donc pas fonctionnel pour d'autres langages.

Pour pouvoir réaliser une analyse sur, par exemple, le langage PHP, il faut donc, comme le montrent [Walden & Doyle & Welch & Whelan, 2009], utiliser un autre outil qui prend en compte les spécificités du langage en question.

C'est pour cela que, pour faire la comparaison entre deux logiciels, il faut tout d'abord définir le but de la comparaison et définir les arguments à comparer.

En ce qui concerne notre comparaison, nous avons décidé de choisir des arguments généraux, qui puissent être utilisables et définissables sur tous les logiciels que nous avons utilisés.

14.9.3. Pouvoir comparer les arguments que l'on souhaite comparer

Du fait de cette multitude d'éléments qu'il est possible de comparer du point de vue de la sécurité entre deux logiciels, notre tableau de comparaison a aussi un intérêt du fait qu'il n'oblige pas à comparer tous les arguments qui y sont définis.

En effet, si le tableau s'étoffe d'arguments au fur et à mesure du temps, il reste toujours possible d'en ignorer certains qui ne sont pas disponibles pour les logiciels à comparer ou tout simplement qui ne font pas partie du but de la comparaison. Pour cela, il suffit simplement de marquer les arguments comme étant « non définissables ».

83 Application Security IDE

Par exemple :

– Si le but de la comparaison de deux logiciels est de procéder à une analyse des données concernant leurs failles, il suffit de rechercher les données pour l'argument en question. Les mesures sur celui-ci seront mises à jour et comparées. Les autres arguments, étant marqués comme « non définissables », ne rentreront pas en compte dans le calcul final pour la comparaison. Il est donc possible de déterminer la sécurité en fonction de certains arguments et pas d'autres.

– De par le même principe, si des éléments ne concernent pas la comparaison de logiciels en question, comme réaliser une analyse sur les injections SQL pour des sites web sans base de données, on pourrait laisser l'argument étant « non définissable » pour ne pas qu'il soit pris en compte.

14.9.4. Ajouter un argument et/ou une mesure de sécurité

Notre tableau de comparaison ne comprenant pas tous les arguments de sécurité existants, il est toujours possible d'en ajouter. En effet, son intérêt tient aussi du fait qu'il n'est pas figé et qu'il est possible de le faire évoluer selon ce que l'on veut comparer.

Pour cela, il faut tout d'abord éditer le fichier « Arguments_Securite_Logiciel-Template.ods ». Il est possible d'ajouter une nouvelle feuille, à l'aide du plus qui se trouve dans les navigations des feuilles. Cela permettra ainsi d'ajouter un nouvel argument.

Pour chaque argument existant, il est possible d'ajouter des nouvelles données à encoder ainsi que des nouvelles mesures qui serviront de comparaison. Pour une meilleure cohérence des données, nous conseillons de définir ces mesures de la même manière que nous avons pu procéder dans ce document.

Une fois cela effectué, il suffit de mettre les liens des nouvelles mesures sur la page « Récapitulatif ».

Une fois que c'est fait, il faut prendre le fichier « Comparaison_Arguments_Securite_Logiciel-Template.ods » pour ajouter les nouvelles mesures de comparaisons. Pour que le fichier puisse aller rechercher les données des logiciels à comparer, il faut ajouter les champs dans la feuille « LIEN_NE_PAS_TOUCHER ».

Par exemple :

Pourcentage de la moyenne des vulnérabilités par rapport aux lignes de code – Poids obtenu	#'Recapitulatif'.B14
Pourcentage de la moyenne des vulnérabilités par rapport aux lignes de code – Non définissable	#'Recapitulatif'.C14
Pourcentage de la moyenne des vulnérabilités par rapport aux lignes de code – Non sécurisé	#'Recapitulatif'.D14

Cela correspond à :

	A	B	C	D
14	Pourcentage de la moyenne des vulnérabilités par rapport aux lignes de code	0,00%	0	0

Ensuite, il suffit d'ajouter les lignes dans la feuille « Comparaison » tout en insérant le nouveau chemin pour accéder aux données.

Par exemple :

C20 | =SI(B7=0; "";INDIRECT(LIEN_NE_PAS_TOUCHER.B7&LIEN_NE_PAS_TOUCHER.B11))

	A	B	C	D	E	F	G	H	I
19	**Arguments de sécurité**	**Logiciel**	**Poids obtenu**	**Non définissable**	**Non sécurisé**	**Point**			
20	Phase de développement du logiciel								
21									
22	Le pourcentage de patchs								

Il est ainsi possible d'obtenir un tableau de comparaison évolutif et adaptable en fonction des comparaisons de logiciels que l'on a besoin de faire.

14.9.5. Limite des données récoltées

Il faut savoir, que, quels que soient les arguments et les mesures définies, il restera toujours des limites dans les résultats obtenus. Ces limites proviennent d'éléments que nous avons déjà évoqués dans le chapitre « *13. Les limites des résultats* » et que nous allons reprendre succinctement ici :

– La publication des données. Est-ce qu'un gestionnaire va publier automatiquement une nouvelle faille détectée, une nouvelle vulnérabilité corrigée ? Est-ce qu'un hacker va prévenir dès qu'il arrive à faire un exploit ?

– La source utilisée. Est-ce qu'elle prend en compte toutes les données existantes ? Les données, prises d'une source différente, donnent-elles le même résultat ?

– L'environnement du logiciel. Il est possible qu'un logiciel, dans certaines conditions, fonctionne parfaitement mais que dans d'autres conditions, pose des problèmes.

– Etc.

15. Conclusion

Le but de ce document était donc d'effectuer une mesure entre la sécurité de logiciels open source et closed source. Cela aurait permis de savoir s'il était possible de déterminer si l'une ou l'autre manière de fournir le code de l'application était plus sécurisée.

Nous avons pour cela défini des critères de sécurité mesurables qui sont les vulnérabilités des logiciels et les exploits réalisés sur les logiciels par des hackers. Nous avons choisi 6 logiciels, 3 open source et 3 closed source, et nous avons comparé les données récoltées.

Le résultat de cette comparaison tendait à montrer que les logiciels en source ouverte étaient plus sécurisés que les logiciels en source fermée.

En réalisant cette comparaison, nous avons pu nous rendre compte que la sécurité d'un logiciel dépendait de beaucoup de facteurs, comme la méthodologie de développement, le temps et l'argent investis par le gestionnaire du logiciel dans son produit, les spécificités du langage de programmation utilisé, etc.

De plus, faire l'analyse de la sécurité n'est pas non plus des plus faciles à réaliser. En effet,, il faudrait pouvoir prendre en compte tous les composants de sécurité existants. Cela n'est pas des plus évidents dû au fait qu'il y en a beaucoup, qu'ils peuvent avoir des définitions différentes et qu'ils ne sont pas tous mesurables.

Pouvoir récolter les données pour mesurer des données de sécurité n'est pas non plus évident. On est dépendant de ce que les personnes[84] veulent bien dire ou non. Ainsi, il est possible qu'un gestionnaire ne publie pas une vulnérabilité détectée, qu'un hacker ne fasse pas part à la communauté de la faille qu'il a su exploiter.

Les données récoltées sont aussi dépendantes de la source utilisée. Des sources différentes peuvent en effet avoir des données différentes qui peuvent varier, par exemple, en fonction de leur critère de sélection. Utiliser une source plutôt qu'une autre peut donc mener à des résultats différents.

C'est pour toutes les raisons énoncées que nous avons réalisé un guide à la décision de logiciel. Celui-ci est composé de deux tableurs contenant les données de deux logiciels à comparer et d'un tableur effectuant la comparaison des données récoltées et déterminant le logiciel le plus sécurisé. Ce guide définit une démarche qui permet d'ajouter, modifier, supprimer des arguments de sécurité mesurable dans le but de pouvoir comparer la sécurité de deux logiciels en fonction de ce que l'on veut analyser. Ce guide n'est dès lors pas figé et peut ainsi évoluer en fonction des différents facteurs cités ci-dessus.

Dans ce guide, nous avons défini de nouveaux critères de sécurité mesurable, ainsi que de

84 Gestionnaire du logiciel, hackers, etc

nouvelles sources à utiliser pour récolter les données d'un logiciel.

Nous avons ainsi à nouveau effectué une analyse de nos 6 logiciels par rapport à ces nouveaux éléments. Le résultat de cette nouvelle analyse ne confirmait pas le résultat obtenu auparavant : les logiciels open source n'étaient plus définis comme étant les plus sécurisés.

Cela prouvait donc bien tout ce que nous avions pu lire auparavant : l'analyse de la sécurité de logiciels n'est pas une science exacte et, en fonction des données utilisées, des arguments comparés, des logiciels étudiés, il est possible d'obtenir des résultats complètement différents.

C'est pour cela que nous pouvons conclure que la sécurité d'un logiciel dépend de beaucoup de facteurs et qu'il y a beaucoup d'éléments à prendre en compte. Ce qui fait qu'il est impossible de conclure qui, de l'open source ou du closed source, et seulement sur cet unique critère, est le plus sécurisé.

Bibliographie

[Schryen & Kadura, 2009] Guido Schryen & Rouven Kadura, Open source vs. Closed source software : towards measuring security, SAC '09, http://dl.acm.org/citation.cfm?id=1529731

[CPE] Common Platform Enumeration, https://cpe.mitre.org/

[Hoepman & Jacobs, 2013] Jaap-Henk Hoepman & Bart Jacobs, Increased security through open source, Communications of the ACM, http://arxiv.org/abs/0801.3924v1

[CVE] Common Vulnerabilities and Exposures, https://cve.mitre.org/

[CVE details] The ultimate security vulnerability datasource, https://www.cvedetails.com/

[CVSS] Common Vulnerabilities Scoring System, https://nvd.nist.gov/cvss.cfm

[CWE] A community-developed dictionary of software weakness types, http://cwe.mitre.org/

[Schryen, 2011] Guido Schryen, Is Open source security a Myth ?, Communications of the ACM, http://dl.acm.org/citation.cfm?id=1941516

[Jonsson & Stömberg & Lindskog, 1999] Erland Jonsson & Lars Strömberg & Stefan Lindskog, On the Functional Relation Between Security and Dependability Impairments, NSP 1999, http://dl.acm.org/citation.cfm?id=335204

[FSF] Free Software Foundation, https://www.fsf.org/

[ISO] International Organization for Standardization, http://www.iso.org/

[NVD] Automating vulnerability managment & security measurement & compliance checking, https://nvd.nist.gov/

[OSI] The Open Source Initiative, http://opensource.org

[USDHS] US Department of Homeland Security, https://www.dhs.gov/

[Walden & Doyle & Welch & Whelan, 2009] James Walden & Maureen Doyle & Grant A. Welch & Michael Whelan, Security of open source web application, ESEM 2009, http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=5314215&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D5314215

[Floyd & Harrington & Hivale, 2007] Kevin Floyd & &Susan J. Harrington & Prachi Hivale, The autotelic propensity of types of hackers, InfoSecCD '07, http://dl.acm.org/citation.cfm?id=1409926&dl=ACM&coll=DL&CFID=581203083&CFTOKEN=87441146

[Furnell, 2010] Steven Furnell, Mac Security : An Apple that can't be bitten ?, Network Security, http://dl.acm.org/citation.cfm?id=2304330

[Seebruck, 2015] Ryan Seebruck, A typology of hackers : Classifuing cyber malfeasance using weighted arc circumplex model, Digital Investigation : The International Journal of Digital Forensics & Incident Response, http://dl.acm.org/citation.cfm?id=2837961

[Colombo & Guerra & Filho & Gomes] Regina Thienne Colombo & Marcelo Schneck Pessöa

& Ana Cervigni Guerra & Amandio Balcao Filho & Célio Caruso Gomes, Prioritization of software security intangible attributes, ACM Sigsoft Software Engineering Notes, http://dl.acm.org/citation.cfm?id=2382781

[Colombo & Guerra & Filho & Gomes] Regina Thienne Colombo & Marcelo Schneck Pessöa & Ana Cervigni Guerra & Amandio Balcao Filho & Célio Caruso Gomes, Prioritization of software security intangible attributes, ACM Sigsoft Software Engineering Notes, http://dl.acm.org/citation.cfm?id=2382781

[Mellado & Fernandez-Medina & Piattini] Daniel Mellado & Eduardo Fernandez-Medina & Mario Piattini, A comparison of software design security metrics, ECSA '10, http://dl.acm.org/citation.cfm?id=1842797

[Morrison] Patrick Morrison, Building a security practices evaluation framework, HotSoS '15, http://dl.acm.org/citation.cfm?id=2746217

[Xie & Lipford & Chu] Jing Xie & Heather Lipford & Bei-Tseng Chu, Evaluating interactuve support for secure programming, CHI '12, http://dl.acm.org/citation.cfm?id=2208665

[Saffiotti & Awyzio & Brown, 2004] Daniel Saffiotti & Gene Awyzio & Robert BK Brown, Myth or Fact : Is open source software more secure than closed source software ?, AUUG 2004, http://ro.uow.edu.au/cgi/viewcontent.cgi?article=1938&context=eispapers

[Alhazmi & Malaiya & &Ray, 2005] Omar Alhazmi & Yashwant Malaiya & Indrajit Ray, Security vulnerabilities in software systems : a quantitative perspective, DBSec '05, http://dl.acm.org/citation.cfm?id=2138953

[Alhazmi & Malaiya & Ray, 2005] Omar Alhazmi & Yashwant Malaiya & Indrajit Ray, Security vulnerabilities in software systems : a quantitative perspective, DBSec '05, http://dl.acm.org/citation.cfm?id=2138953

[Exploit DB] Offensive security exploit database archive, https://www.exploit-db.com/

[Offensive Security] Offensive security, manager of exploit db, https://www.offensive-security.com/

[Microsoft] Microsoft, https://www.microsoft.com

[ITSecDB] Vunerability, patch and compliance datasource, http://www.itsecdb.com/

UNIVERSITÉ DE NAMUR
Faculté d'informatique
Année académique 2015-2016

Logiciel open source VS closed source : mesure de la sécurité

Annexes

David Hostaux

Promoteur : _____ (Signature pour approbation du dépôt - REE art. 40)

Jean-Noël Colin

Mémoire présenté en vue de l'obtention du grade de
Master en Sciences Informatiques.

Table des matières

1. Les termes à ne pas confondre....9
1.1. Conclusions :....9
2. Le choix des logiciels....11
2.1. Le noyau Linux....11
2.2. Windows 7....11
2.3. Aucun logiciel Apple....11
2.4. Microsoft, l'entreprise des logiciels fermés....12
3. Lorsqu'il y a peu de données....13
3.1. Les sites des logiciels....13
3.2. [Exploit DB]....13
4. Les totaux ne correspondent pas....14
5. Failles du noyau Linux....15
5.1. Liste des vulnérabilités....15
5.1.1. 2015....15
5.1.2. 2014....17
5.1.3. 2013....20
5.1.4. 2012....23
5.1.5. 2011....26
5.2. Récapitulatif....27
5.2.1. 2015....27
5.2.2. 2014....28
5.2.3. 2013....28
5.2.4. 2012....28
5.2.5. 2011....28
6. Failles de Windows 7....29
6.1. Liste des vulnérabilités....29
6.1.1. 2015....29
6.1.2. 2014....32

6.1.3. 2013....33
6.1.4. 2012....35
6.1.5. 2011....36
6.2. Récapitulatif....38
6.2.1. 2015....38
6.2.2. 2014....38
6.2.3. 2013....38
6.2.4. 2012....39
6.2.5. 2011....39
7. Failles de Libre Office....40
7.1. Liste des vulnérabilités....40
7.1.1. 2015....40
7.1.2. 2014....40
7.1.3. 2013....41
7.1.4. 2012....41
7.1.5. 2011....41
7.2. Récapitulatif....41
7.2.1. 2015....41
7.2.2. 2014....41
7.2.3. 2013....41
7.2.4. 2012....42
7.2.5. 2011....42
8. Failles de Microsoft Office....43
8.1. Liste des vulnérabilités....43
8.1.1. 2015....43
8.1.2. 2014....44
8.1.3. 2013....44
8.1.4. 2012....45
8.1.5. 2011....45
8.2. Récapitulatif....46

8.2.1. 2015....46
8.2.2. 2014....46
8.2.3. 2013....46
8.2.4. 2012....46
8.2.5. 2011....47
9. Failles d'Apache HTTP Serveur....48
9.1. Liste des vulnérabilités....48
9.1.1. 2008 - 2015....48
10. Failles d'IIS Serveur....51
10.1. Liste des vulnérabilités....51
10.1.1. 2008 – 2015....51
11. Les exploits vérifiés....53
12. Exploits sur le noyau Linux....54
12.1. Liste des exploits....54
12.2. Récapitulatif....55
13. Exploits sur Windows 7....56
13.1. Liste des exploits....56
13.2. Récapitulatif....57
14. Exploits sur Libre Office....58
14.1. Liste des exploits....58
14.2. Récapitulatif....58
15. Comparaison sur Microsoft Office....59
15.1. Liste des exploits....59
15.2. Récapitulatif....60
16. Exploits sur Apache HTTP Serveur....61
16.1. Liste des exploits....61
16.2. Récapitulatif....61
17. Exploits sur IIS Serveur....62
17.1. Liste des exploits....62
17.2. Récapitulatif....62

18. Comparaison des exploits entre les 6 logiciels.....63
19. Failles et exploits du noyau Linux.....64
19.1. Les failles CVE-Details.....64
19.2. Analyse des exploits par rapport aux failles CVE-Details.....64
20. Failles et exploits de Windows 7.....66
20.1. Les failles CVE-Details.....66
20.2. Analyse des exploits par rapport aux failles CVE-Details.....66
21. Récapitulatif des données de comparaison entre le noyau Linux et Windows 7.....68
22. Failles et exploits de Libre Office.....69
22.1. Les failles CVE-Details.....69
22.2. Analyse des exploits par rapport aux failles CVE-Details.....69
23. Failles et exploits de Microsoft Office.....70
23.1. Les failles CVE-Details.....70
23.2. Analyse des exploits par rapport aux failles CVE-Details.....70
24. Récapitulatif des données de comparaison entre Libre Office et Microsoft Office.....72
25. Failles et exploits d'Apache HTTP Serveur.....73
25.1. Les failles CVE-Details.....73
25.2. Analyse des exploits par rapport aux failles CVE-Details.....73
26. Failles et exploits de IIS Serveur.....74
26.1. Les failles CVE-Details.....74
26.2. Analyse des exploits par rapport aux failles CVE-Details.....74
27. Récapitulatif des données de comparaison entre Apache HTTP Serveur et IIS Serveur.....75
28. Récapitulatif des données de comparaison entre les logiciels open source et closed source.....76
29. Analyse des données du noyau Linux.....78
29.1. SE – Phase de développement du logiciel.....78
29.2. SE – Le pourcentage de patchs réalisés.....78
29.3. SE – Les exploits réalisés sur un logiciel.....78
29.4. SI – Le poids des failles d'un logiciel.....79
29.4.1. Détails.....79
29.5. SI – Les lignes de code du logiciel.....79

30. Analyse des données de Windows 7....80
30.1. SE – Phase de développement du logiciel....80
30.2. SE – Le pourcentage de patchs réalisés....80
30.3. SE – Les exploits réalisés sur un logiciel....80
30.4. SI – Le poids des failles d'un logiciel....81
30.4.1. Détails....81
30.5. SI – Les lignes de code du logiciel....81
31. Comparaison noyau Linux et Windows 7....82
32. Analyse des données de Libre Office....83
32.1. SE – Phase de développement du logiciel....83
32.2. SE – Le pourcentage de patchs réalisés....83
32.3. SE – Les exploits réalisés sur un logiciel....83
32.4. SI – Le poids des failles d'un logiciel....84
32.4.1. Détails....84
32.5. SI – Les lignes de code du logiciel....84
33. Analyse des données de Microsoft Office....85
33.1. SE – Phase de développement du logiciel....85
33.2. SE – Le pourcentage de patchs réalisés....85
33.3. SE – Les exploits réalisés sur un logiciel....85
33.4. SI – Le poids des failles d'un logiciel....86
33.4.1. Détails....86
33.5. SI – Les ligne de code du logiciel....86
34. Comparaison Libre Office et Microsoft Office....87
35. Analyse des données d'Apache HTTP Serveur....88
35.1. SE – Phase de développement du logiciel....88
35.2. SE – Le pourcentage de patchs réalisés....88
35.3. SE – Les exploits réalisés sur un logiciel....88
35.4. SI – Le poids des failles d'un logiciel....89
35.4.1. Détails....89
35.5. SI – Les lignes de code du logiciel....89

36. Analyse des données d'IIS Serveur.....90
36.1. Phase de développement du logiciel.....90
36.2. SE – Le pourcentage de patchs réalisés.....90
36.3. SE – Les exploits réalisés sur un logiciel.....90
36.4. SI – Le poids des failles d'un logiciel.....91
36.4.1. Détails.....91
36.5. SI – Les lignes de code du logiciel.....91
37. Comparaison Apache HTTP Serveur et IIS Serveur.....92

1. Les termes à ne pas confondre

Bien souvent, ces différents termes sont utilisés et confondus les uns avec les autres alors qu'ils veulent exprimer différentes choses. Voici un récapitulatif de leur signification [GNU] :

– **Logiciel en source ouverte :** Tout le monde peut accéder au code source source du programme, le redistribuer et en faire des travaux dérivés.

– **Logiciel en source fermée :** L'accès au code source n'est pas disponible à tout le monde. Bien souvent, seuls ceux qui travaillent sur ce logiciel ont accès au code.

– **Logiciel libre :** L'utilisateur est libre de faire ce qu'il veut du programme : l'utiliser, l'étudier, le modifier, redistribuer des copies (modifiées ou non).

– **Logiciel propriétaire :** Ne permet pas, légalement ou techniquement, tout type d'utilisation, l'étude du code source, la distribution de copies ou la modification du code source.

– **Logiciel commercial :** Logiciel destiné à la vente.

– **Logiciel gratuit :** Logiciel qu'il est possible d'obtenir gratuitement.

1.1. Conclusions :

On peut donc déduire ([GNU] , [FSF]) de ces différentes définitions :

– Même si le logiciel open source et le logiciel libre désignent plus ou moins la même catégorie de logiciel, ils n'en restent pas moins différents. La source ouverte désigne une méthodologie de développement (enjeux pratiques, performance) tandis que le logiciel libre désigne un mouvement de société (impératif éthique, respect de la liberté de l'utilisateur). L'open source défend dès lors des valeurs plus faibles que celles du logiciel libre. Ce qui veut dire que tous les logiciels libres répondent aux critères de l'open source mais qu'un logiciel open source ne répond pas nécessairement à tous les critères du logiciel libre. Exemple : des exécutables open source mais dont l'utilisateur ne peut pas faire fonctionner de version modifiée : les critères de l'open source (qui s'intéresse uniquement à la licence du code source) sont respectés mais l'exécutable n'est pas libre.

– Le logiciel closed source et le logiciel propriétaire représentent aussi la même catégorie de logiciel. Les logiciels en source fermée ont des licences propriétaires et donc, des droits d'auteurs ou copyright associés.

– Un logiciel open source / libre, tout comme un logiciel closed source / propriétaire, peut être commercial ou gratuit.

– Avec l'anglais « free » qui peut avoir deux significations en français : « libre » et « gratuit », il faut pouvoir distinguer si l'on veut parler de liberté ou de prix lorsque l'on voit l’appellation « Free software ».

2. Le choix des logiciels

2.1. Le noyau Linux

Le noyau Linux n'est à proprement parler pas un système d'exploitation en tant que tel. Nous avons décidé d'en utiliser les données disponibles car tous les systèmes Linux[1] partent de ce noyau pour développer leur système.

Comme le démontrent [Alhazmi & Malaiya & Ray, 2005], lorsque l'on utilise un même élément de logiciel pour passer d'une version à l'autre (dans ce cas-ci, d'une distribution à l'autre), les vulnérabilités sont elles aussi passées entre les versions.

En analysant directement les données du noyau Linux plutôt que d'une distribution en particulier, on analyse donc les failles qui ont le potentiel de se retrouver sur plusieurs distributions Linux différentes.

2.2. Windows 7

Il existe plusieurs systèmes d'exploitation [Microsoft][2]. Nous avons choisi d'analyser les données de Windows 7 tout simplement car il est sorti en 2009 et qu'il est toujours utilisé aujourd'hui. Il existe donc plus de données qu'il est possible d'exploiter et comparer.

De plus, [Alhazmi & Malaiya & Ray, 2005] montrent qu'il existe un cycle au niveau des systèmes d'exploitation Windows du point de vue des failles :

– Phase 1 : Le système commence à être déployé, les utilisateurs commencent à l'utiliser et les premiers efforts concernant la résolution de vulnérabilités doivent être réalisés.

– Phase 2 : Le système commence à avoir son pic de popularité, il est le plus utilisé et l'effort concernant les vulnérabilités est à son maximum.

– Phase 3 : Le système commence à être remplacé par un autre. Les utilisateurs commencent à le quitter et la fréquence des patchs diminue de même que le support.

En choisissant Windows 7, nous étudions un système d'exploitation qui parcourt ces 3 phases de vie et donc, ces 3 étapes par rapport à la lutte contre les failles[3].

2.3. Aucun logiciel Apple

Vous l'aurez remarqué, nous n'avons analysé aucun logiciel d'Apple. Cela s'explique par le fait que, comme on peut le voir sur leur site, [Apple OS], contrairement à ce que certains peuvent penser, distribue des produits « open source ». Cependant, les licences open source utilisées ne sont pas les licences standards faites par l'Open-Source Initiative [OSI] et/ou maintenues par la Free

1 Debian, Red Hat, Ubuntu, CentOS, etc
2 Le dernier en date : Windows 10, mais il y a aussi Windows 8, Windows 7, Windows Vista, etc
3 La phase 3 pour Windows 7 commence en 2013 avec l'arrivée de Windows 8.

Software Foundation [FSF] mais bien des licences développées et maintenues par Apple lui-même[4].

Comme a pu le montrer [Furnell, 2010], il faut faire attention à la communication faite par Apple, où le message que l'entreprise diffuse n'est pas toujours vrai ou nécessairement le bon. Il aurait donc fallu, avant de choisir un logiciel Apple pour faire une comparaison quelconque, s'assurer que les licences utilisées sont bien celles qu'elles prétendent être et que les logiciels sont aussi open source qu'ils le font croire.

L'analyse de la communication d'Apple ne faisant pas l'objet de ce mémoire, et ne voulant pas perdre de temps avec cela, nous avons préféré ne pas choisir de logiciel de cette marque. Cela nous évite ainsi de les classer, peut-être à mauvais escient, dans la catégorie open source ou closed source.

2.4. Microsoft, l'entreprise des logiciels fermés

[Microsoft], fournit ses logiciels en sources fermées. L'entreprise a d'ailleurs cette image qui lui colle à la peau.

Cependant, certains signes montrent qu'il pourrait y avoir un changement de politique de l'entreprise avec la « libération » de leur framework [.NET] qui est maintenant passé en open source.

Mais les logiciels que nous avons choisis sont, eux, toujours bel et bien disponibles en source fermée.

4 Par exemple : la licence du système d'exploitation de Mac OS X est « Apple's Open Source Licence »

3. Lorsqu'il y a peu de données

Comme vous pourrez le remarquer, pour certains logiciels tout comme pour certaines années, il y a peu d'informations disponibles. Pour être certains de ne pas passer à côté d'informations utiles, nous avons voulu récolter d'autres informations à partir d'autres sources.

3.1. Les sites des logiciels

Les logiciels open source fournissent la plupart du temps un lien qui regroupe les différentes vulnérabilités publiées comme par exemple [vulnerabilités libre office] ou encore [vulnérabilités apache http]. Nous n'avons pas utilisé ces différentes sources car les vulnérabilités qui y sont décrites sont reprises sur [CVE details].

De plus, il est difficile de trouver ces mêmes informations (aussi clairement formulées et accessibles) sur les sites des logiciels closed source étudiés. [Hoepman & Jacobs, 2013] expliquent cela par le fait que lorsqu'une entreprise publie ce genre d'informations, elle peut perdre de la crédibilité et donc, le respect de ses utilisateurs. Pour éviter cela, elles ne rendent pas accessibles ces informations.

3.2. [Exploit DB]

Nous n'avons pas non plus utilisé ce site comme source de notre première analyse car la plupart des vulnérabilités publiées sur ce site le sont aussi sur [CVE details]. De plus, comme cette base de données est alimentée par des utilisateurs externes, nous nous en servirons dans la partie consacrée aux hackers. Par conséquent, il sera donc possible de croiser les informations recueillies sur [Exploit DB] et [CVE details].

4. Les totaux ne correspondent pas

Il est possible de se retrouver avec des totaux qui ne correspondent pas vraiment à la somme réelle des éléments.

Par exemple :

	2015	2014	2013	2012	2011	Total
Noyau Linux	*551*	*526*	*487*	*503*	*513*	2579

Si l'on calcule le total réel : 551 + 526 + 487 + 503 + 513, on obtient 2580 et non 2579.

Bien que cela semble être une erreur, ceci s'explique par le fait que pour obtenir les données par année, nous avons arrondi au plus près. Le total ne s'effectue pas par rapport aux données arrondies mais bien par la somme des données de bases qui sera arrondie au plus près.

Par exemple :

– Prenons deux chiffres : 2,7 + 1,6. Arrondi au plus près, on obtient 3 + 2 égal 5. Mais en calculant le total par rapport aux nombres de base : 2,7+1,6, le résultat est 4,3. Arrondi au plus près, le total vaut 4 et non 5. Les résultats ne sont pas les mêmes, mais les données restent néanmoins correctes.

Maintenant, la marge de différence ne grimpera pas et restera toujours de + ou – 1. Ce que nous pouvons considérer comme une différence raisonnable.

5. Failles du noyau Linux

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2011	83	62	1	21	10					1	21	9			1
2012	115	83	4	24	10					6	19	11			
2013	189	101	6	41	13					11	57	26			6
2014	133	89	8	21	10					11	30	20			8
2015	77	47	6	12	2					11	10	16			

Source : http://www.cvedetails.com/product/47/Linux-Linux-Kernel.html?vendor_id=33

Date de dernière consultation : 29/02/2016

5.1. Liste des vulnérabilités

5.1.1. 2015

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1							*1*			
2		*1*								
3							*1*			
4		*1*								
5					*1*					
6					*1*					
7						*1*				
8		*1*								
9		*1*								
10		*1*								
11					*1*					
12				*1*						
13					*1*					
14					*1*					
15		*1*								

16					*1*					
17								*1*		
18					*1*					
19					*1*					
20							*1*			
21					*1*					
22					*1*					
23					*1*					
24							*1*			
25									*1*	
26								*1*		
27									*1*	
28									*1*	
29					*1*					
30						*1*				
31					*1*					
32									*1*	
33		*1*								
34							*1*			
35							*1*			
36					*1*					
37							*1*			
38			*1*							
39		*1*								
40							*1*			
41					*1*					
42					*1*					
43							*1*			
44					*1*					
45								*1*		
46										*1*
47		*1*								
48					*1*					
49								*1*		
50								*1*		
51					*1*					
52								*1*		
53							*1*			
54										*1*
55							*1*			
56					*1*					
57		*1*								
58					*1*					
59					*1*					
60					*1*					
61					*1*					
62							*1*			
63				*1*						
64		*1*								
65		*1*								

66		*1*								
67							*1*			
68								*1*		
69							*1*			
70					*1*					
71					*1*					
72							*1*			
73							*1*			
74							*1*			
75							*1*			
76					*1*					
77		*1*								

Source : http://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/year-2015/Linux-Linux-Kernel.html

5.1.2. 2014

Numéro failles\poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1					*1*					
2		*1*								
3							*1*			
4					*1*					
5					*1*					
6						*1*				
7					*1*					
8					*1*					
9					*1*					
10					*1*					
11					*1*					
12		*1*								
13		*1*								
14					*1*					
15					*1*					
16					*1*					
17					*1*					
18					*1*					
19					*1*					
20					*1*					
21					*1*					
22						*1*				
23					*1*					
24					*1*					
25								*1*		
26							*1*			
27								*1*		
28					*1*					
29					*1*					
30				*1*						
31				*1*						
32						*1*				
33							*1*			

34					*1*					
35						*1*				
36							*1*			
37							*1*			
38					*1*					
39					*1*					
40					*1*					
41					*1*					
42							*1*			
43					*1*					
44					*1*					
45					*1*					
46					*1*					
47								*1*		
48							*1*			
49					*1*					
50					*1*					
51		*1*								
52						*1*				
53					*1*					
54				*1*						
55			*1*							
56					*1*					
57					*1*					
58								*1*		
59								*1*		
60		*1*								
61		*1*								
62		*1*								
63							*1*			
64					*1*					
65					*1*					
66				*1*						
67								*1*		
68							*1*			
69							*1*			
70							*1*			
71					*1*					
72							*1*			
73							*1*			
74							*1*			
75							*1*			
76					*1*					
77					*1*					
78					*1*					
79					*1*					
80							*1*			
81					*1*					
82							*1*			
83					*1*					
84					*1*					
85							*1*			
86										*1*

87		*1*								
88			*1*							
89										*1*
90						*1*				
91					*1*					
92				*1*						
93				*1*						
94		*1*								
95					*1*					
96							*1*			
97			*1*							
98		*1*								
99		*1*								
100		*1*								
101					*1*					
102		*1*								
103							*1*			
104					*1*					
105							*1*			
106		*1*								
107						*1*				
108			*1*							
109					*1*					
110							*1*			
111									*1*	
112						*1*				
113						*1*				
114							*1*			
115							*1*			
116					*1*					
117					*1*					
118					*1*					
119					*1*					
120					*1*					
121					*1*					
122					*1*					
123					*1*					
124					*1*					
125					*1*					
126					*1*					
127					*1*					
128			*1*							
129					*1*					
130					*1*					
131								*1*		
132					*1*					
133					*1*					

Source: http://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/year-2014/Linux-Linux-Kernel.html

5.1.3. 2013

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1						*1*				
2					*1*					
3							*1*			
4					*1*					
5					*1*					
6							*1*			
7				*1*						
8							*1*			
9					*1*					
10				*1*						
11					*1*					
12						*1*				
13						*1*				
14							*1*			
15				*1*						
16				*1*						
17						*1*				
18							*1*			
19							*1*			
20				*1*						
21							*1*			
22					*1*					
23					*1*					
24					*1*					
25					*1*					
26					*1*					
27							*1*			
28					*1*					
29							*1*			
30						*1*				
31					*1*					
32							*1*			
33						*1*				
34							*1*			
35							*1*			
36						*1*				
37				*1*						
38							*1*			
39								*1*		
40					*1*					
41					*1*					
42					*1*					
43					*1*					
44					*1*					
45					*1*					
46					*1*					
47				*1*						
48							*1*			

49					*1*					
50					*1*					
51					*1*					
52					*1*					
53					*1*					
54					*1*					
55					*1*					
56					*1*					
57					*1*					
58					*1*					
59					*1*					
60					*1*					
61					*1*					
62					*1*					
63					*1*					
64					*1*					
65					*1*					
66				*1*						
67			*1*							
68					*1*					
69		*1*								
70					*1*					
71					*1*					
72					*1*					
73					*1*					
74					*1*					
75					*1*					
76					*1*					
77					*1*					
78					*1*					
79						*1*				
80							*1*			
81						*1*				
82								*1*		
83		*1*								
84		*1*								
85		*1*								
86							*1*			
87		*1*								
88		*1*								
89		*1*								
90		*1*								
91		*1*								
92					*1*					
93					*1*					
94		*1*								
95		*1*								
96		*1*								
97					*1*					
98		*1*								
99				*1*						
100					*1*					

101							*1*			
102					*1*					
103								*1*		
104					*1*					
105							*1*			
106				*1*						
107		*1*								
108					*1*					
109		*1*								
110							*1*			
111				*1*						
112					*1*					
113							*1*			
114							*1*			
115						*1*				
116							*1*			
117						*1*				
118						*1*				
119					*1*					
120						*1*				
121							*1*			
122							*1*			
123					*1*					
124				*1*						
125						*1*				
126				*1*						
127						*1*				
128							*1*			
129								*1*		
130				*1*						
131							*1*			
132								*1*		
133							*1*			
134		1								
135			1							
136						1				
137							1			
138							1			
139					1					
140					1					
141						1				
142					1					
143						1				
144					1					
145					1					
146					1					
147		1								
148		1								
149		1								
150		1								
151		1								
152		1								

153		1								
154		1								
155		1								
156		1								
157		1								
158		1								
159		1								
160		1								
161		1								
162				1						
163				1						
164					1					
165		1								
166		1								
167					1					
168					1					
169				1						
170							1			
171					1					
172							1			
173							1			
174				1						
175		1								
176				1						
177				1						
178					1					
179						1				
180							1			
181						1				
182					1					
183					1					
184								1		
185					1					
186			1							
187				1						
188								1		
189		1								

Source : http://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/year-2013/Linux-Linux-Kernel.html

5.1.4. 2012

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1					*1*					
2				*1*						
3					*1*					
4		*1*								
5							*1*			
6					*1*					
7					*1*					
8		*1*								

9						*1*				
10						*1*				
11		*1*								
12								*1*		
13								*1*		
14					*1*					
15					*1*					
16								*1*		
17		*1*								
18					*1*					
19					*1*					
20					*1*					
21					*1*					
22				*1*						
23							*1*			
24	*1*									
25							*1*			
26				*1*						
27					*1*					
28							*1*			
29					*1*					
30							*1*			
31					*1*					
32					*1*					
33					*1*					
34										*1*
35							*1*			
36					*1*					
37					*1*					
38					*1*					
39								*1*		
40					*1*					
41							*1*			
42					*1*					
43							*1*			
44					*1*					
45							*1*			
46						*1*				
47								*1*		
48					*1*					
49					*1*					
50					*1*					
51								*1*		
52							*1*			
53							*1*			
54					*1*					
55					*1*					
56		*1*								
57					*1*					
58					*1*					
59					*1*					
60		*1*								

61					*1*					
62					*1*					
63					*1*					
64				*1*						
65							*1*			
66					*1*					
67					*1*					
68					*1*					
69					*1*					
70					*1*					
71										*1*
72							*1*			
73					*1*					
74					*1*					
75					*1*					
76					*1*					
77								*1*		
78							*1*			
79					*1*					
80					*1*					
81							*1*			
82					*1*					
83		*1*								
84		*1*								
85		*1*								
86							*1*			
87		*1*								
88		*1*								
89		*1*								
90		*1*								
91				*1*						
92							*1*			
93					*1*					
94			*1*							
95					*1*					
96					*1*					
97						*1*				
98					*1*					
99								*1*		
100					*1*					
101					*1*					
102				*1*						
103		*1*								
104		*1*								
105		*1*								
106					*1*					
107		*1*								
108					*1*					
109				*1*						
110					*1*					
111		*1*								
112					*1*					

113			*1*							
114					*1*					
115					*1*					

Source : http://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/year-2012/Linux-Linux-Kernel.html

5.1.5. 2011

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1					*1*					
2						*1*				
3		*1*								
4					*1*					
5					*1*					
6				*1*						
7								*1*		
8		*1*								
9					*1*					
10					*1*					
11								*1*		
12							*1*			
13							*1*			
14						*1*				
15					*1*					
16								*1*		
17					*1*					
18					*1*					
19							*1*			
20							*1*			
21					*1*					
22					*1*					
23					*1*					
24					*1*					
25						*1*				
26							*1*			
27							*1*			
28						*1*				
29					*1*					
30		*1*								
31		*1*								
32		*1*								
33							*1*			
34		*1*								
35								*1*		
36					*1*					
37					*1*					
38					*1*					
39								*1*		
40		*1*								
41		*1*								
42							*1*			

43							*1*			
44							*1*			
45					*1*					
46					*1*					
47					*1*					
48		*1*								
49						*1*				
50						*1*				
51		*1*								
52		*1*								
53								*1*		
54						*1*				
55							*1*			
56							*1*			
57		*1*								
58						*1*				
59					*1*					
60						*1*				
61	*1*									
62							*1*			
63		*1*								
64							*1*			
65							*1*			
66		*1*								
67						*1*				
68					*1*					
69						*1*				
70					*1*					
71				*1*						
72					*1*					
73								*1*		
74					*1*					
75					*1*					
76							*1*			
77		*1*								
78		*1*								
79		*1*								
80								*1*		
81							*1*			
82				*1*						
83					*1*					

Source : http://www.cvedetails.com/vulnerability-list/vendor_id-33/product_id-47/year-2011/Linux-Linux-Kernel.html

5.2. Récapitulatif

5.2.1. 2015

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombre de failles	*0*	*14*	*1*	*2*	*27*	*2*	*18*	*7*	*4*	*2*

Nombre total de failles :	77
Poids total	424
Poids en %:	**551**

5.2.2. 2014

Poids	1	2	3	4	5	6	7	8	9	10
Nombre de failles	*0*	*14*	*5*	*6*	*65*	*9*	*24*	*7*	*1*	*2*

Nombre total de failles : 133
Poids total 699
Poids en %: **525,6**

5.2.3. 2013

Poids	1	2	3	4	5	6	7	8	9	10
Nombre de failles	*0*	*35*	*3*	*21*	*70*	*20*	*33*	*7*	*0*	*0*

Nombre total de failles : 189
Poids total 920
Poids en %: **486,8**

5.2.4. 2012

Poids	1	2	3	4	5	6	7	8	9	10
Nombre de failles	*1*	*18*	*2*	*7*	*56*	*4*	*17*	*8*	*0*	*2*

Nombre total de failles : 115
Poids total 578
Poids en %: **502,6**

5.2.5. 2011

Poids	1	2	3	4	5	6	7	8	9	10
Nombre de failles	*1*	*17*	*0*	*3*	*26*	*11*	*17*	*8*	*0*	*0*

Nombre total de failles : 83
Poids total 426
Poids en %: **513,3**

6. Failles de Windows 7

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2011	102	14	18	9	8		2			4	2	65			3
2012	44	4	14	6						2	3	22			
2013	100	16	19	24	6			1		3	2	67			4
2014	36	6	12	5	3					6	5	12			4
2015	147	11	52	12	9			1		24	24	60			1

Source : http://www.cvedetails.com/product/17153/Microsoft-Windows-7.html?vendor_id=26

Date de dernière consultation : 29/02/2016

6.1. Liste des vulnérabilités

6.1.1. 2015

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1							*1*			
2							*1*			
3							*1*			
4							*1*			
5									*1*	
6									*1*	
7							*1*			
8				*1*						
9							*1*			
10				*1*						
11						*1*				
12									*1*	
13									*1*	
14									*1*	
15									*1*	
16		*1*								

17							*1*			
18							*1*			
19							*1*			
20									*1*	
21					*1*					
22					*1*					
23							*1*			
24							*1*			
25							*1*			
26									*1*	
27							*1*			
28									*1*	
29							*1*			
30									*1*	
31							*1*			
32							*1*			
33				*1*						
34									*1*	
35									*1*	
36									*1*	
37							*1*			
38							*1*			
39									*1*	
40							*1*			
41									*1*	
42							*1*			
43			*1*							
44									*1*	
45				*1*						
46		*1*								
47									*1*	
48									*1*	
49									*1*	
50									*1*	
51									*1*	
52									*1*	
53									*1*	
54									*1*	
55									*1*	
56		*1*								
57					*1*					
58									*1*	
59		*1*								
60									*1*	
61									*1*	
62									*1*	
63		*1*								
64									*1*	
65				*1*						
66					*1*					
67					*1*					
68							*1*			

69		*1*								
70										*1*
71							*1*			
72							*1*			
73							*1*			
74							*1*			
75		*1*								
76							*1*			
77							*1*			
78							*1*			
79							*1*			
80							*1*			
81							*1*			
82							*1*			
83									*1*	
84							*1*			
85							*1*			
86							*1*			
87							*1*			
88							*1*			
89							*1*			
90							*1*			
91							*1*			
92		*1*								
93					*1*					
94							*1*			
95							*1*			
96									*1*	
97									*1*	
98									*1*	
99									*1*	
100									*1*	
101		*1*								
102		*1*								
103		*1*								
104		*1*								
105		*1*								
106		*1*								
107									*1*	
108									*1*	
109							*1*			
110							*1*			
111				*1*						
112										*1*
113							*1*			
114									*1*	
115						*1*				
116		*1*								
117									*1*	
118									*1*	
119									*1*	
120									*1*	
121					*1*					

122									*1*	
123					*1*					
124		*1*								
125									*1*	
126				*1*						
127								*1*		
128		*1*								
129				*1*						
130							*1*			
131				*1*						
132							*1*			
133							*1*			
134				1						
135					1					
136							1			
137							1			
138									1	
139										1
140					1					
141		1								
142			1							
143								1		
144						1				
145							1			
146							1			
147							1			

Source :

http://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-17153/year-2015/Microsoft-Windows-7.html

6.1.2. 2014

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1					*1*					
2									*1*	
3									*1*	
4									*1*	
5				*1*						
6										*1*
7					*1*					
8							*1*			
9									*1*	
10									*1*	
11									*1*	
12							*1*			
13									*1*	
14					*1*					
15							*1*			
16								*1*		
17							*1*			
18									*1*	
19							*1*			
20									*1*	

21									*1*	
22							*1*			
23							*1*			
24					*1*					
25							*1*			
26							*1*			
27							*1*			
28							*1*			
29							*1*			
30							*1*			
31									*1*	
32							*1*			
33					*1*					
34							*1*			
35									*1*	
36							*1*			

Source :

http://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-17153/year-2014/Microsoft-Windows-7.html

6.1.3. 2013

Numéro failles\poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1					*1*					
2									*1*	
3									*1*	
4									*1*	
5							*1*			
6							*1*			
7								*1*		
8									*1*	
9							*1*			
10					*1*					
11							*1*			
12							*1*			
13							*1*			
14					*1*					
15					*1*					
16							*1*			
17							*1*			
18							*1*			
19							*1*			
20					*1*					
21							*1*			
22							*1*			
23							*1*			
24							*1*			
25							*1*			
26										*1*
27								*1*		
28										*1*
29									*1*	
30							*1*			

31					*1*					
32							*1*			
33							*1*			
34							*1*			
35				*1*						
36									*1*	
37									*1*	
38								*1*		
39								*1*		
40							*1*			
41							*1*			
42							*1*			
43							*1*			
44							*1*			
45							*1*			
46							*1*			
47									*1*	
48							*1*			
49							*1*			
50							*1*			
51							*1*			
52					*1*					
53							*1*			
54							*1*			
55							*1*			
56							*1*			
57							*1*			
58							*1*			
59							*1*			
60							*1*			
61							*1*			
62							*1*			
63					*1*					
64					*1*					
65					*1*					
66					*1*					
67					*1*					
68					*1*					
69					*1*					
70					*1*					
71					*1*					
72					*1*					
73					*1*					
74					*1*					
75					*1*					
76					*1*					
77					*1*					
78					*1*					
79					*1*					
80					*1*					
81					*1*					
82					*1*					

83					*1*					
84					*1*					
85					*1*					
86					*1*					
87					*1*					
88					*1*					
89					*1*					
90					*1*					
91					*1*					
92					*1*					
93							*1*			
94								*1*		
95									*1*	
96						*1*				
97										*1*
98							*1*			
99									*1*	
100										*1*

Source :

http://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-17153/year-2013/Microsoft-Windows-7.html

6.1.4. 2012

Numéro failles\poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1										*1*
2									*1*	
3										*1*
4									*1*	
5							*1*			
6					*1*					
7							*1*			
8							*1*			
9							*1*			
10							*1*			
11							*1*			
12				*1*						
13							*1*			
14							*1*			
15							*1*			
16							*1*			
17										*1*
18					*1*					
19									*1*	
20									*1*	
21									*1*	
22							*1*			
23										*1*
24							*1*			
25							*1*			
26							*1*			
27									*1*	

28		*1*								
29									*1*	
30									*1*	
31							*1*			
32				*1*						
33							*1*			
34				*1*						
35									*1*	
36									*1*	
37							*1*			
38									*1*	
39									*1*	
40									*1*	
41									*1*	
42									*1*	
43				*1*						
44						*1*				

Source :

http://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-17153/year-2012/Microsoft-Windows-7.html

6.1.5. 2011

Numéro failles\poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1									*1*	
2							*1*			
3				*1*						
4									*1*	
5									*1*	
6							*1*			
7								*1*		
8							*1*			
9									*1*	
10									*1*	
11									*1*	
12							*1*			
13									*1*	
14									*1*	
15										*1*
16							*1*			
17									*1*	
18							*1*			
19									*1*	
20					*1*					
21									*1*	
22							*1*			
23									*1*	
24					*1*					
25							*1*			
26							*1*			
27				*1*						
28							*1*			
29							*1*			

30							*1*			
31							*1*			
32							*1*			
33							*1*			
34							*1*			
35							*1*			
36							*1*			
37							*1*			
38							*1*			
39							*1*			
40							*1*			
41							*1*			
42									*1*	
43								*1*		
44								*1*		
45					*1*					
46							*1*			
47							*1*			
48							*1*			
49										*1*
50								*1*		
51								*1*		
52							*1*			
53									*1*	
54							*1*			
55							*1*			
56							*1*			
57							*1*			
58							*1*			
59							*1*			
60							*1*			
61							*1*			
62							*1*			
63							*1*			
64							*1*			
65							*1*			
66							*1*			
67							*1*			
68							*1*			
69							*1*			
70							*1*			
71							*1*			
72							*1*			
73							*1*			
74							*1*			
75							*1*			
76							*1*			
77							*1*			
78							*1*			
79							*1*			
80							*1*			
81							*1*			
82							*1*			

83										*1*
84									*1*	
85									*1*	
86								*1*		
87				*1*						
88						*1*				
89							*1*			
90							*1*			
91							*1*			
92							*1*			
93									*1*	
94									*1*	
95									*1*	
96									*1*	
97							*1*			
98									*1*	
99								*1*		
100								*1*		
101								*1*		
102							*1*			

Source :

http://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-17153/year-2011/Microsoft-Windows-7.html

6.2. Récapitulatif

6.2.1. 2015

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombre de failles	*0*	*18*	*2*	*10*	*10*	*3*	*53*	*2*	*46*	*3*

Nombre total de failles : 147
Poids total 981
Poids en %: **667,3**

6.2.2. 2014

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombre de failles	*0*	*0*	*0*	*1*	*5*	*0*	*16*	*1*	*12*	*1*

Nombre total de failles : 36
Poids total 267
Poids en %: **741,7**

6.2.3. 2013

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombre de failles	*0*	*0*	*0*	*1*	*37*	*1*	*42*	*5*	*10*	*4*

Nombre total de failles : 100
Poids total 659
Poids en %: **659**

6.2.4. 2012

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombre de failles	*0*	*1*	*0*	*4*	*2*	*1*	*17*	*0*	*15*	*4*

Nombre total de failles : 44
Poids total 328
Poids en %: **745,5**

6.2.5. 2011

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombre de failles	*0*	*0*	*0*	*3*	*3*	*1*	*62*	*9*	*21*	*3*

Nombre total de failles : 102
Poids total 758
Poids en %: **743,1**

7. Failles de Libre Office

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2011	2	1	1	2											
2012	5	4	3	3							1				
2014	3	2	2												
2015	5	4	4	3	3						1				

Source : https://www.cvedetails.com/product/21008/Libreoffice-Libreoffice.html?vendor_id=11439

Date de dernière consultation : 29/02/2016

7.1. Liste des vulnérabilités

7.1.1. 2015

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1							*1*			
2							*1*			
3							*1*			
4				*1*						
5							*1*			

Source : https://www.cvedetails.com/vulnerability-list/vendor_id-11439/product_id-21008/year-2015/Libreoffice-Libreoffice.html

7.1.2. 2014

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1								*1*		
2								*1*		
3										*1*

Source : https://www.cvedetails.com/vulnerability-list/vendor_id-11439/product_id-21008/year-2014/Libreoffice-Libreoffice.html

7.1.3. 2013

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10

Source :

https://www.cvedetails.com/vulnerability-list/vendor_id-11439/product_id-21008/year-2013/Libreoffice-Libreoffice.html

7.1.4. 2012

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1				*1*						
2								*1*		
3							*1*			
4								*1*		
5				*1*						

Source :

https://www.cvedetails.com/vulnerability-list/vendor_id-11439/product_id-21008/year-2012/Libreoffice-Libreoffice.html

7.1.5. 2011

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1				*1*						
2									*1*	

Source :

https://www.cvedetails.com/vulnerability-list/vendor_id-11439/product_id-21008/year-2011/Libreoffice-Libreoffice.html

7.2. Récapitulatif

7.2.1. 2015

Poids	1	2	3	4	5	6	7	8	9	10
Nombre de failles	*0*	*0*	*0*	*1*	*0*	*0*	*4*	*0*	*0*	*0*

Nombre total de failles : 5
Poids total 32
Poids en %: **640**

7.2.2. 2014

Poids	1	2	3	4	5	6	7	8	9	10
Nombre de failles	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*2*	*0*	*1*

Nombre total de failles : 3
Poids total 26
Poids en %: **866,7**

7.2.3. 2013

Poids	1	2	3	4	5	6	7	8	9	10
Nombre de failles	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*

Nombre total de failles : 0
Poids total 0
Poids en %: **0**

7.2.4. 2012

Poids	1	2	3	4	5	6	7	8	9	10
Nombre de failles	*0*	*0*	*0*	*2*	*0*	*0*	*1*	*2*	*0*	*0*

Nombre total de failles : 5
Poids total 31
Poids en %: **620**

7.2.5. 2011

Poids	1	2	3	4	5	6	7	8	9	10
Nombre de failles	*0*	*0*	*0*	*1*	*0*	*0*	*0*	*0*	*1*	*0*

Nombre total de failles : 2
Poids total 13
Poids en %: **650**

8. Failles de Microsoft Office

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2011	30	12	28	17	14							2			
2012	19	3	16	6	6							2			
2013	17	3	13	8	5						3	1			1
2014	10	2	5	1	1					2	1	1			
2015	40	6	37	19	23		1			1	1	1			

Source : https://www.cvedetails.com/product/320/Microsoft-Office.html?vendor_id=26

Date de dernière consultation : 29/02/2016

8.1. Liste des vulnérabilités

8.1.1. 2015

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1									*1*	
2									*1*	
3									*1*	
4									*1*	
5									*1*	
6									*1*	
7									*1*	
8									*1*	
9									*1*	
10									*1*	
11									*1*	
12									*1*	
13									*1*	
14									*1*	
15									*1*	

16									*1*	
17									*1*	
18									*1*	
19									*1*	
20									*1*	
21									*1*	
22									*1*	
23				*1*						
24									*1*	
25									*1*	
26									*1*	
27									*1*	
28									*1*	
29									*1*	
30									*1*	
31									*1*	
32									*1*	
33									*1*	
34									*1*	
35				*1*						
36									*1*	
37									*1*	
38									*1*	
39									*1*	
40				*1*						

Source : https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-320/year-2015/Microsoft-Office.html

8.1.2. 2014

Numéro failles\poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1									*1*	
2									*1*	
3									*1*	
4									*1*	
5					*1*					
6							*1*			
7				*1*						
8									*1*	
9									*1*	
10									*1*	

Source : https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-320/year-2014/Microsoft-Office.html

8.1.3. 2013

Numéro failles\poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1				*1*						
2				*1*						
3									*1*	
4									*1*	
5							*1*			
6									*1*	
7									*1*	

8					*1*					
9									*1*	
10									*1*	
11									*1*	
12									*1*	
13									*1*	
14					*1*					
15									*1*	
16									*1*	
17									*1*	

Source : https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-320/year-2013/Microsoft-Office.html

8.1.4. 2012

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1				*1*						
2									*1*	
3									*1*	
4							*1*			
5									*1*	
6									*1*	
7									*1*	
8							*1*			
9									*1*	
10									*1*	
11									*1*	
12									*1*	
13									*1*	
14									*1*	
15									*1*	
16									*1*	
17									*1*	
18									*1*	
19									*1*	

Source : https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-320/year-2012/Microsoft-Office.html

8.1.5. 2011

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10
1									*1*	
2									*1*	
3									*1*	
4									*1*	
5									*1*	
6									*1*	
7									*1*	
8									*1*	
9									*1*	
10									*1*	
11									*1*	
12									*1*	
13									*1*	

14									*1*	
15									*1*	
16									*1*	
17									*1*	
18									*1*	
19									*1*	
20									*1*	
21									*1*	
22									*1*	
23									*1*	
24									*1*	
25									*1*	
26									*1*	
27									*1*	
28									*1*	
29									*1*	
30									*1*	

Source : https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-320/year-2011/Microsoft-Office.html

8.2. Récapitulatif

8.2.1. 2015

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombre de failles	*0*	*0*	*0*	*3*	*0*	*0*	*0*	*0*	*37*	*0*

Nombre total de failles :	40
Poids total	345
Poids en %:	**863**

8.2.2. 2014

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombre de failles	*0*	*0*	*0*	*1*	*1*	*0*	*1*	*0*	*7*	*0*

Nombre total de failles :	10
Poids total	79
Poids en %:	**790**

8.2.3. 2013

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombre de failles	*0*	*0*	*0*	*2*	*2*	*0*	*1*	*0*	*12*	*0*

Nombre total de failles :	17
Poids total	133
Poids en %:	**782,4**

8.2.4. 2012

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombre de failles	*0*	*0*	*0*	*1*	*0*	*0*	*2*	*0*	*16*	*0*

Nombre total de failles :	19
Poids total	162
Poids en %:	**852,6**

8.2.5. 2011

Poids	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Nombre de failles	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*30*	*0*

Nombre total de failles :	30
Poids total	270
Poids en %:	**900**

9. Failles d'Apache HTTP Serveur

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2008	12	2			1		6		1			1	1		
2009	8	5								1		1			1
2010	9	3	2	1			1				3				1
2011	12	8		1								1			2
2012	8	4		1			1				2	1			
2013	5	1	1				2								
2014	11	9	1	2						2	1				1
2015	4	2								1					

Source : https://www.cvedetails.com/product/66/Apache-Http-Server.html?vendor_id=45

Date de dernière consultation : 02/03/2016

9.1. Liste des vulnérabilités

9.1.1. 2008 - 2015

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10	Année
1				*1*							2015
2					*1*						2015
3					*1*						2015
4					*1*						2015
5				*1*							2014
6					*1*						2014
7					*1*						2014
8					*1*						2014
9							*1*				2014
10				*1*							2014
11				*1*							2014
12					*1*						2014

13					*1*						2014
14					*1*						2014
15				*1*							2014
16								*1*			2013
17				*1*							2013
18					*1*						2013
19				*1*							2013
20				*1*							2013
21					*1*						2012
22				*1*							2012
23			*1*								2012
24					*1*						2012
25							*1*				2012
26				*1*							2012
27					*1*						2012
28			*1*								2012
29	*1*										2011
30				*1*							2011
31				*1*							2011
32				*1*							2011
33					*1*						2011
34				*1*							2011
35								*1*			2011
36				*1*							2011
37				*1*							2011
38					*1*						2011
39				*1*							2011
40					*1*						2011
41					*1*						2010
42					*1*						2010
43					*1*						2010
44				*1*							2010
45										*1*	2010
46					*1*						2010
47							*1*				2010
48			*1*								2010
49				*1*							2010
50						*1*					2009
51								*1*			2009
52			*1*								2009
53					*1*						2009
54								*1*			2009
55							*1*				2009
56							*1*				2009
57					*1*						2009
58				*1*							2008
59							*1*				2008
60					*1*						2008
61				*1*							2008
62			*1*								2008
63				*1*							2008
64				*1*							2008

65								*1*			2008
66				*1*							2008
67				*1*							2008
68				*1*							2008
69				*1*							2008

Sources :

https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/year-2015/Apache-Http-Server.html

https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/year-2014/Apache-Http-Server.html

https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/year-2013/Apache-Http-Server.html

https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/year-2012/Apache-Http-Server.html

https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/year-2011/Apache-Http-Server.html

https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/year-2010/Apache-Http-Server.html

https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/year-2009/Apache-Http-Server.html

https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-66/year-2008/Apache-Http-Server.html

10. Failles d'IIS Serveur

Year	# of Vulnerabilities	DoS	Code Execution	Overflow	Memory Corruption	Sql Injection	XSS	Directory Traversal	Http Response Splitting	Bypass something	Gain Information	Gain Privileges	CSRF	File Inclusion	# of exploits
2008	4	1	1	1								1			
2009	7	1	1	1	1					3					2
2010	6	2	3	3	1		1			1	1				1
2012	1										1				
2013	1														
2014	1														
2015	1														
Total	21	4	5	5	2		1			4	2	1			3

Source : https://www.cvedetails.com/product/3436/Microsoft-IIS.html?vendor_id=26

Date de dernière consultation : 02/03/2016

10.1. Liste des vulnérabilités

10.1.1. 2008 – 2015

Numéro failles\poids	1	2	3	4	5	6	7	8	9	10	Année
1				*1*							2015
2						*1*					2014
3				*1*							2013
4		*1*									2012
5										*1*	2010
6							*1*				2010
7									*1*		2010
8				*1*							2010
9									*1*		2010
10			*1*								2010
11						*1*					2009
12						*1*					2009
13						*1*					2009

14								*1*		2009
15			*1*							2009
16							*1*			2009
17							*1*			2009
18									*1*	2008
19					*1*					2008
20								*1*		2008
21							*1*			2008

Sources :

https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-3436/year-2015/Microsoft-IIS.html

https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-3436/year-2014/Microsoft-IIS.html

https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-3436/year-2013/Microsoft-IIS.html

https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-3436/year-2012/Microsoft-IIS.html

https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-3436/year-2011/Microsoft-IIS.html

https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-3436/year-2010/Microsoft-IIS.html

https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-3436/year-2009/Microsoft-IIS.html

https://www.cvedetails.com/vulnerability-list/vendor_id-26/product_id-3436/year-2008/Microsoft-IIS.html

11. Les exploits vérifiés

Comme le précise [Exploit DB], ce n'est pas parce qu'une attaque n'est pas vérifiée que cela veut dire qu'elle n'existe pas. Cela signifie seulement dire qu'[Offensive Security] a pris la peine d'effectuer la vérification.

Ainsi, une attaque non validée veut donc tout simplement dire qu'[Offensive Security] n'a pas pris le temps ou eu l'opportunité de faire les tests nécessaires pour reproduire l'exploit.

Nous allons quand même prêter notre attention par rapport à la réalisation de cette vérification car cela permet d'obtenir un deuxième point de vue sur l'analyse. De ce fait, nous apprendrons si cet organisme préfère vérifier les exploits des logiciels open source ou des logiciels closed source.

12. Exploits sur le noyau Linux

12.1. Liste des exploits

Posté	Vérifié	Date	Source
1		2015-10-14	https://www.exploit-db.com/exploits/38454/
1		2015-04-13	https://www.exploit-db.com/exploits/36743/
	1	2015-03-09	https://www.exploit-db.com/exploits/36310/
1		2015-03-04	https://www.exploit-db.com/exploits/36266/
1		2015-03-04	https://www.exploit-db.com/exploits/36267/
1		2015-03-04	https://www.exploit-db.com/exploits/36268/
1		2014-11-25	https://www.exploit-db.com/exploits/35370/
1		2014-10-09	https://www.exploit-db.com/exploits/34923/
1		2014-07-21	https://www.exploit-db.com/exploits/34134/
	1	2014-06-21	https://www.exploit-db.com/exploits/33824/
	1	2014-05-28	https://www.exploit-db.com/exploits/39214/
1		2014-05-26	https://www.exploit-db.com/exploits/33516/
1		2014-02-11	https://www.exploit-db.com/exploits/31574/
	1	2014-02-02	https://www.exploit-db.com/exploits/31346/
	1	2014-02-02	https://www.exploit-db.com/exploits/31347/
1		2014-01-31	https://www.exploit-db.com/exploits/31305/
	1	2013-12-10	https://www.exploit-db.com/exploits/38826/
1		2013-08-02	https://www.exploit-db.com/exploits/27297/
	1	2013-06-11	https://www.exploit-db.com/exploits/26131/
	1	2013-06-07	https://www.exploit-db.com/exploits/38559/
	1	2013-06-05	https://www.exploit-db.com/exploits/38589/
1		2013-05-14	https://www.exploit-db.com/exploits/25444/
1		2013-05-14	https://www.exploit-db.com/exploits/25450/
	1	2013-04-15	https://www.exploit-db.com/exploits/38465/
1		2013-03-13	https://www.exploit-db.com/exploits/24747/
	1	2013-03-13	https://www.exploit-db.com/exploits/38390/
	1	2013-02-24	https://www.exploit-db.com/exploits/33336/
1		2013-02-05	https://www.exploit-db.com/exploits/24459/
	1	2012-12-13	https://www.exploit-db.com/exploits/38132/
	1	2012-10-09	https://www.exploit-db.com/exploits/37937/
	1	2012-07-26	https://www.exploit-db.com/exploits/37543/
	1	2012-07-19	https://www.exploit-db.com/exploits/19933/
	1	2012-07-05	https://www.exploit-db.com/exploits/19605/
	1	2012-01-23	https://www.exploit-db.com/exploits/18411/
	1	2012-01-12	https://www.exploit-db.com/exploits/35161/
	1	2012-01-03	https://www.exploit-db.com/docs/18304.pdf
	1	2011-12-29	https://www.exploit-db.com/exploits/36545/
	1	2011-12-06	https://www.exploit-db.com/exploits/36425/
	1	2011-11-07	https://www.exploit-db.com/exploits/36294/
1		2011-11-04	https://www.exploit-db.com/exploits/18080/
	1	2011-09-05	https://www.exploit-db.com/exploits/17787/
1		2011-09-01	https://www.exploit-db.com/exploits/17769/
	1	2011-06-02	https://www.exploit-db.com/exploits/35820/

	1	2011-04-11	https://www.exploit-db.com/exploits/35600/
1		2011-03-10	https://www.exploit-db.com/exploits/16952/
	1	2011-03-02	https://www.exploit-db.com/exploits/16263/
	1	2011-03-02	https://www.exploit-db.com/exploits/35403/
	1	2011-03-02	https://www.exploit-db.com/exploits/35404/
1		2011-01-10	https://www.exploit-db.com/exploits/15962/
1		2011-01-08	https://www.exploit-db.com/exploits/15944/
	1	2011-01-05	https://www.exploit-db.com/exploits/15916/
	1	2010-12-18	https://www.exploit-db.com/exploits/15774/
	1	2010-12-07	https://www.exploit-db.com/exploits/15704/
	1	2010-11-29	https://www.exploit-db.com/docs/15634.pdf
	1	2010-11-27	https://www.exploit-db.com/exploits/15622/
1		2010-11-26	https://www.exploit-db.com/exploits/15619/
	1	2010-11-24	https://www.exploit-db.com/exploits/35013/
1		2010-11-10	https://www.exploit-db.com/exploits/15481/
	1	2010-11-09	https://www.exploit-db.com/exploits/34987/
	1	2010-10-28	https://www.exploit-db.com/exploits/15344/
	1	2010-10-19	https://www.exploit-db.com/exploits/15285/
	1	2010-09-29	https://www.exploit-db.com/exploits/15150/
	1	2010-09-16	https://www.exploit-db.com/exploits/15023/
	1	2010-09-16	https://www.exploit-db.com/exploits/15024/
	1	2010-08-27	https://www.exploit-db.com/exploits/14814/
	1	2010-08-09	https://www.exploit-db.com/exploits/14594/
	1	2010-05-18	https://www.exploit-db.com/exploits/34001/
	1	2010-04-27	https://www.exploit-db.com/exploits/33886/
	1	2010-04-09	https://www.exploit-db.com/exploits/12130/
	1	2010-02-02	https://www.exploit-db.com/exploits/33592/
	1	2010-02-01	https://www.exploit-db.com/exploits/33585/

12.2. Récapitulatif

	Postés	Vérifiés	Total	Pourcentage Vérification
Total	23	48	71	67.61%

13. Exploits sur Windows 7

13.1. Liste des exploits

Posté	Vérifié	Date	Source
	1	2015-11-23	https://www.exploit-db.com/exploits/38793/
	1	2015-09-24	https://www.exploit-db.com/exploits/38307/
	1	2015-09-22	https://www.exploit-db.com/exploits/38266/
	1	2015-09-22	https://www.exploit-db.com/exploits/38267/
	1	2015-09-22	https://www.exploit-db.com/exploits/38276/
	1	2015-09-22	https://www.exploit-db.com/exploits/38277/
	1	2015-09-22	https://www.exploit-db.com/exploits/38278/
	1	2015-09-22	https://www.exploit-db.com/exploits/38279/
	1	2015-09-22	https://www.exploit-db.com/exploits/38280/
	1	2015-08-21	https://www.exploit-db.com/exploits/37922/
1		2015-08-13	https://www.exploit-db.com/exploits/37768/
1		2015-08-12	https://www.exploit-db.com/exploits/37755/
	1	2014-03-24	https://www.exploit-db.com/exploits/32477/
	1	2013-09-23	https://www.exploit-db.com/exploits/28482/
	1	2013-06-03	https://www.exploit-db.com/exploits/25912/
	1	2013-01-29	https://www.exploit-db.com/exploits/24437/
	1	2012-06-13	https://www.exploit-db.com/exploits/37396/
	1	2012-04-25	https://www.exploit-db.com/exploits/18780/
	1	2011-12-24	https://www.exploit-db.com/exploits/18271/
	1	2011-11-22	https://www.exploit-db.com/exploits/36327/
	1	2011-10-13	https://www.exploit-db.com/exploits/17978/
	1	2011-09-05	https://www.exploit-db.com/exploits/17783/
	1	2011-03-31	https://www.exploit-db.com/exploits/35553/
	1	2011-03-17	https://www.exploit-db.com/docs/16994.pdf
	1	2011-01-23	https://www.exploit-db.com/docs/16032.pdf
	1	2011-01-02	https://www.exploit-db.com/exploits/15894/
	1	2010-12-21	https://www.exploit-db.com/exploits/15803/
	1	2010-11-24	https://www.exploit-db.com/exploits/15609/
1		2010-08-24	https://www.exploit-db.com/exploits/14733/
	1	2010-08-20	https://www.exploit-db.com/exploits/14697/
	1	2010-08-17	https://www.exploit-db.com/exploits/14666/
	1	2010-08-17	https://www.exploit-db.com/exploits/14667/
	1	2010-08-17	https://www.exploit-db.com/exploits/14670/
1		2010-08-11	https://www.exploit-db.com/exploits/14613/
	1	2010-06-01	https://www.exploit-db.com/exploits/13729/
	1	2010-05-28	https://www.exploit-db.com/exploits/13719/
	1	2010-05-07	https://www.exploit-db.com/exploits/12524/
	1	2010-04-17	https://www.exploit-db.com/exploits/12273/
	1	2010-03-28	https://www.exploit-db.com/docs/11921.pdf
	1	2010-03-01	https://www.exploit-db.com/exploits/13631/
	1	2010-02-28	https://www.exploit-db.com/exploits/13630/
	1	2010-02-22	https://www.exploit-db.com/exploits/11531/

13.2. Récapitulatif

	Postés	Vérifiés	Total	Pourcentage Vérification
Total	4	38	42	90.48%

14. Exploits sur Libre Office

14.1. Liste des exploits

Posté	Vérifié	Date	Source
	1	2012-05-28	https://www.exploit-db.com/exploits/18940/
1		2012-04-19	https://www.exploit-db.com/exploits/18754/

14.2. Récapitulatif

	Postés	Vérifiés	Total	Pourcentage Vérification
Total	1	1	2	50.00%

15. Comparaison sur Microsoft Office

15.1. Liste des exploits

Posté	Vérifié	Date	Source
	1	2015-12-14	https://www.exploit-db.com/exploits/39233/
	1	2015-12-09	https://www.exploit-db.com/exploits/38968/
	1	2015-09-16	https://www.exploit-db.com/exploits/38918/
	1	2015-09-16	https://www.exploit-db.com/exploits/38214/
	1	2015-09-16	https://www.exploit-db.com/exploits/38215/
	1	2015-09-16	https://www.exploit-db.com/exploits/38216/
1		2015-08-31	https://www.exploit-db.com/exploits/38217/
	1	2018-08-25	https://www.exploit-db.com/exploits/38031/
	1	2015-08-25	https://www.exploit-db.com/exploits/37967/
	1	2015-08-21	https://www.exploit-db.com/exploits/37909/
	1	2015-08-21	https://www.exploit-db.com/exploits/37910/
	1	2015-08-21	https://www.exploit-db.com/exploits/37911/
	1	2015-08-21	https://www.exploit-db.com/exploits/37912/
	1	2015-08-21	https://www.exploit-db.com/exploits/37913/
	1	2015-08-21	https://www.exploit-db.com/exploits/37924/
	1	2015-02-28	https://www.exploit-db.com/exploits/36207/
1		2014-11-12	https://www.exploit-db.com/exploits/35216/
	1	2013-07-01	https://www.exploit-db.com/exploits/26517/
	1	2013-02-20	https://www.exploit-db.com/exploits/24526/
	1	2012-11-20	https://www.exploit-db.com/exploits/22850/
	1	2012-11-09	https://www.exploit-db.com/exploits/22591/
	1	2012-10-29	https://www.exploit-db.com/exploits/22330/
	1	2012-10-28	https://www.exploit-db.com/exploits/22310/
	1	2012-10-25	https://www.exploit-db.com/exploits/22237/
	1	2012-10-24	https://www.exploit-db.com/exploits/22215/
	1	2012-10-11	https://www.exploit-db.com/exploits/37980/
	1	2012-07-31	https://www.exploit-db.com/exploits/20122/
	1	2010-06-11	https://www.exploit-db.com/exploits/19037/
	1	2012-01-08	https://www.exploit-db.com/exploits/18334/
	1	2011-11-22	https://www.exploit-db.com/exploits/18143/
	1	2011-11-05	https://www.exploit-db.com/exploits/18087/
	1	2011-07-03	https://www.exploit-db.com/exploits/17474/
	1	2011-06-26	https://www.exploit-db.com/exploits/17451/
	1	2011-06-14	https://www.exploit-db.com/exploits/17399/
	1	2011-04-29	https://www.exploit-db.com/exploits/17227/
	1	2010-10-16	https://www.exploit-db.com/exploits/15262/
	1	2010-09-11	https://www.exploit-db.com/exploits/14971/
	1	2010-09-11	https://www.exploit-db.com/docs/14972.pdf
	1	2010-09-08	https://www.exploit-db.com/exploits/14944/
	1	2010-09-08	https://www.exploit-db.com/docs/14946.pdf
1		2010-08-25	https://www.exploit-db.com/exploits/14746/
1		2010-08-25	https://www.exploit-db.com/exploits/14782/
1		2010-04-06	https://www.exploit-db.com/exploits/12079/

15.2. Récapitulatif

	Postés	Vérifiés	Total	Pourcentage Vérification
Total	5	38	43	88.37%

16. Exploits sur Apache HTTP Serveur

16.1. Liste des exploits

Posté	Vérifié	Date	Source
	1	2013-10-04	https://www.exploit-db.com/exploits/28713/
	1	2012-02-06	https://www.exploit-db.com/exploits/36663/
	1	2012-01-31	https://www.exploit-db.com/exploits/18442/
1		2011-12-09	https://www.exploit-db.com/exploits/18221/
	1	2011-11-24	https://www.exploit-db.com/exploits/36352/
1		2011-08-19	https://www.exploit-db.com/exploits/17696/

16.2. Récapitulatif

	Postés	Vérifiés	Total	Pourcentage Vérification
Total	2	4	6	66.67%

17. Exploits sur IIS Serveur

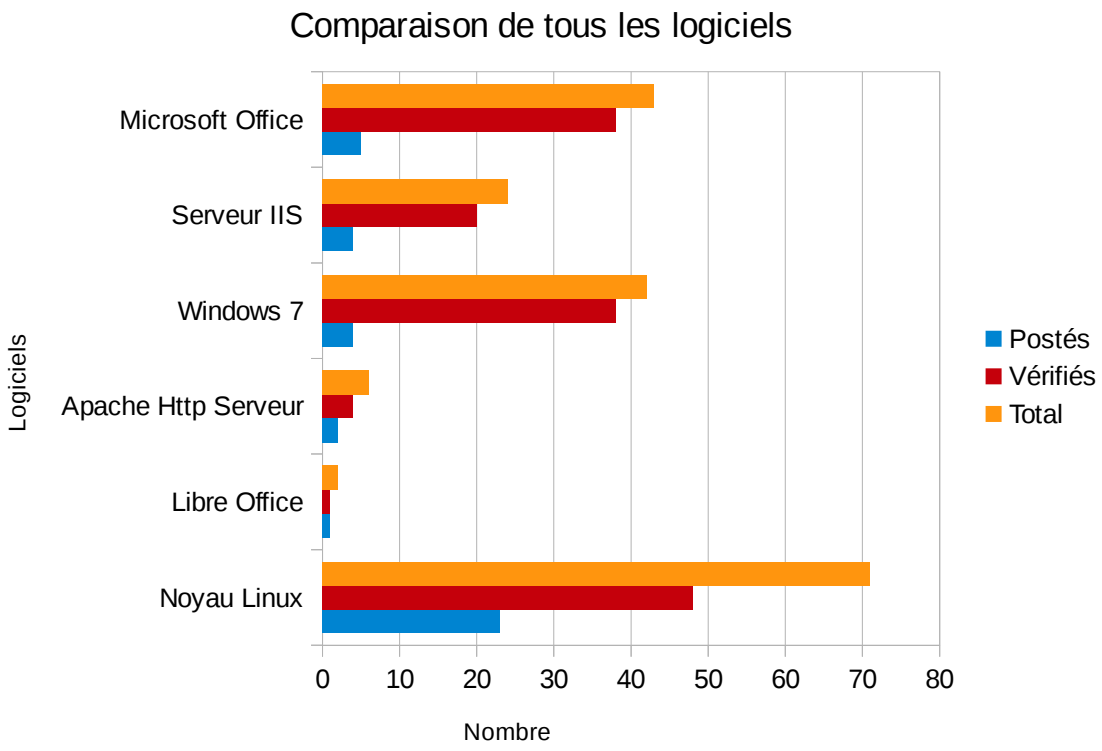
17.1. Liste des exploits

Posté	Vérifié	Date	Source
1		2015-08-31	https://www.exploit-db.com/docs/38033.pdf
1		2014-09-29	https://www.exploit-db.com/exploits/34817/
	1	2012-07-02	https://www.exploit-db.com/exploits/19525/
	1	2012-07-02	https://www.exploit-db.com/docs/19527.pdf
1		2012-06-10	https://www.exploit-db.com/exploits/19033/
	1	2012-06-08	https://www.exploit-db.com/exploits/19026/
1		2011-07-03	https://www.exploit-db.com/exploits/17476/
	1	2011-01-08	https://www.exploit-db.com/exploits/16467/
	1	2010-12-21	https://www.exploit-db.com/exploits/15803/
	1	2010-11-12	https://www.exploit-db.com/exploits/16740/
	1	2010-10-01	https://www.exploit-db.com/exploits/15167/
	1	2010-09-20	https://www.exploit-db.com/exploits/16358/
	1	2010-09-20	https://www.exploit-db.com/exploits/16471/
	1	2010-07-25	https://www.exploit-db.com/exploits/16355/
	1	2010-07-25	https://www.exploit-db.com/exploits/16356/
	1	2010-07-25	https://www.exploit-db.com/exploits/16470/
	1	2010-07-07	https://www.exploit-db.com/exploits/16354/
	1	2010-07-02	https://www.exploit-db.com/exploits/14179/
	1	2010-06-15	https://www.exploit-db.com/exploits/13886/
	1	2010-06-15	https://www.exploit-db.com/exploits/16472/
	1	2010-04-30	https://www.exploit-db.com/exploits/16357/
	1	2010-04-30	https://www.exploit-db.com/exploits/16468/
	1	2010-04-30	https://www.exploit-db.com/exploits/16469/
	1	2010-02-13	https://www.exploit-db.com/papers/13620/

17.2. Récapitulatif

	Postés	Vérifiés	Total	Pourcentage Vérification
Total	4	20	24	83.33%

18. Comparaison des exploits entre les 6 logiciels



19. Failles et exploits du noyau Linux

19.1. Les failles CVE-Details

2010	124
2011	83
2012	115
2013	189
2014	133
2015	77

19.2. Analyse des exploits par rapport aux failles CVE-Details

Date	Listé	Non Listé	Poids	Sans poids	CVE
2015-10-14		*1*	*10*		2015-3036
2015-04-13	*1*		*7*		2014-7822
2015-03-09		*1*		*1*	MAL FORMATE
2015-03-04	*1*		*7*		2014-9322
2015-03-04	*1*		*7*		2014-4943
2015-03-04	*1*		*7*		2014-3631
2014-11-25	*1*		*7*		2014-3153
2014-10-09	*1*		*6*		2014-5207
2014-07-21	*1*		*7*		2014-4699
2014-06-21	*1*		*6*		2014-4014
2014-05-28	*1*		*2*		2014-1739
2014-05-26	*1*		*7*		2014-0196
2014-02-11	*1*		*7*		2013-6282
2014-02-02	*1*		*7*		2014-0038
2014-02-02	*1*		*7*		2014-0038
2014-01-31	*1*		*7*		2014-0038
2013-12-10	*1*		*4*		2013-4579
2013-08-02	*1*		*6*		2013-0268
2013-06-11	*1*		*7*		2013-2094
2013-06-07	*1*		*7*		2013-2852
2013-06-05		*1*		*1*	NON COMMUNIQUE
2013-05-14	*1*		*7*		2013-2094
2013-05-14	*1*		*4*		2013-1959
2013-04-15	*1*		*7*		2013-3301
2013-03-13	*1*		*7*		2013-1828
2013-03-13		*1*		*1*	NON COMMUNIQUE
2013-02-24	*1*		*7*		2013-1763
2013-02-05	*1*		*2*		2013-0160
2012-12-13	*1*		*4*		2012-5375
2012-10-09	*1*		*5*		2012-0957
2012-07-26	*1*		*2*		2012-3430
2012-07-19		*1*	*7*		2009-2692
2012-07-05	*1*		*5*		2012-3375

2012-01-23	*1*		*7*		2012-0056
2012-01-12		*1*		*1*	NON COMMUNIQUE
2012-01-03		*1*		*1*	NON COMMUNIQUE
2011-12-29	*1*		*5*		2012-0045
2011-12-06	*1*		*8*		2011-2189
2011-11-07		*1*		*1*	NON COMMUNIQUE
2011-11-04		*1*		*1*	NON COMMUNIQUE
2011-09-05	*1*		*2*		2012-4073
2011-09-01	*1*		*5*		2011-2918
2011-06-02	*1*		*4*		2011-2183
2011-04-11	*1*		*5*		2011-1479
2011-03-10	*1*		*5*		2010-4165
2011-03-02	*1*		*5*		2010-4165
2011-03-02	*1*		*5*		2011-1083
2011-03-02	*1*		*5*		2011-1082
2011-01-10		*1*		*1*	MAL FORMATE
2011-01-08		*1*		*1*	NON COMMUNIQUE
2011-01-05		*1*		*1*	NON COMMUNIQUE
2010-12-18	*1*		*7*		2010-4347
2010-12-07	*1*		*6*		2010-4258
2010-11-29		*1*		*1*	NON COMMUNIQUE
2010-11-27	*1*		*5*		2010-4249
2010-11-26	*1*		*5*		2010-3858
2010-11-24	*1*		*5*		2010-4250
2010-11-10		*1*		*1*	NON COMMUNIQUE
2010-11-09	*1*		*2*		2010-4158
2010-10-28	*1*		*6*		2010-2963
2010-10-19	*1*		*7*		2010-3904
2010-09-29	*1*		*7*		2010-3437
2010-09-16	*1*		*7*		2010-3301
2010-09-16	*1*		*7*		2010-3081
2010-08-27	*1*		*7*		2010-2959
2010-08-09		*1*		*1*	NON COMMUNIQUE
2010-05-18	*1*		*2*		2010-1636
2010-04-27	*1*		*2*		2010-1437
2010-04-09	*1*		*7*		2010-1146
2010-02-02		*1*		*1*	NON COMMUNIQUE
2010-02-01	*1*		*5*		2010-0307

20. Failles et exploits de Windows 7

20.1. Les failles CVE-Details

2010	64
2011	102
2012	44
2013	100
2014	36
2015	147

20.2. Analyse des exploits par rapport aux failles CVE-Details

Date	Listé	Non Listé	Poids	Sans poids	CVE
2015-11-23	*1*		*7*		2015-6098
2015-09-24	*1*		*7*		2015-2512
2015-09-22	*1*		*7*		2015-2366
2015-09-22	*1*		*7*		2015-2365
2015-09-22	*1*		*7*		2015-2511
2015-09-22	*1*		*7*		2015-2518
2015-09-22	*1*		*7*		2015-2517
2015-09-22	*1*		*7*		2015-2507
2015-09-22	*1*		*7*		2015-2512
2015-08-21	*1*		*9*		2015-2459
2015-08-13	*1*		*7*		2015-2370
2015-08-12		*1*	*7*		2014-4076
2014-03-24		*1*	*7*		2014-2671
2013-09-23		*1*	*9*		2013-0810
2013-06-03		*1*		*1*	MAL FORMATE
2013-01-29		*1*		*1*	NON COMMUNIQUE
2012-06-13		*1*		*1*	NON COMMUNIQUE
2012-04-25		*1*	*9*		2012-0158
2011-12-24		*1*		*1*	NON COMMUNIQUE
2011-11-22		*1*		*1*	NON COMMUNIQUE
2011-10-13	*1*		*9*		2011-2003
2011-09-05		*1*		*1*	NON COMMUNIQUE
2011-03-31		*1*		*1*	NON COMMUNIQUE
2011-03-17		*1*		*1*	NON COMMUNIQUE
2011-01-23	*1*		*7*		2010-1893
2011-01-02	*1*		*7*		2010-1744
2010-12-21		*1*	*10*		2010-3972
2010-11-24	*1*		*7*		2010-4398
2010-08-24	*1*		*9*		2010-3143
2010-08-20		*1*		*1*	NON COMMUNIQUE
2010-08-17	*1*		*7*		2010-1888
2010-08-17	*1*		*7*		2010-1889
2010-08-17	*1*		*5*		2010-1890

2010-08-11		*1*		*1*	NON COMMUNIQUE
2010-06-01		*1*		*1*	NON COMMUNIQUE
2010-05-28		*1*		*1*	NON COMMUNIQUE
2010-05-07		*1*	*10*		2009-3103
2010-04-17	*1*		*10*		2010-0269
2010-03-28		*1*		*1*	NON COMMUNIQUE
2010-03-01		*1*		*1*	NON COMMUNIQUE
2010-02-28		*1*		*1*	NON COMMUNIQUE
2010-02-22		*1*	*4*		2010-07-18

21. Récapitulatif des données de comparaison entre le noyau Linux et Windows 7

	Noyau Linux	Windows 7
Exploits listés	*55*	*20*
Exploits non listés	*16*	*22*
Nombre total d'exploits	*71*	*42*
Exploits sans poids	*14*	*15*
Poids des exploits	*325*	*203*
Poids/100 exploits	*570*	*752*
Failles CVE-details	*721*	*493*
Exploits non listés	*16*	*22*
Total failles	*737*	*515*
Nombre de failles	*737*	*515*
Nombre d'exploits	*71*	*42*
Pourcentage	*9.63%*	*8.16%*

Tableau récapitulatif des données entre le noyau Linux et Windows 7

	Noyau Linux	Windows 7
Failles CVE-details	721	493
Exploits non listés	16	22
Total failles	737	515

Tableau récapitulatif des failles CVE et des exploits non listés entre le noyau Linux et Windows 7

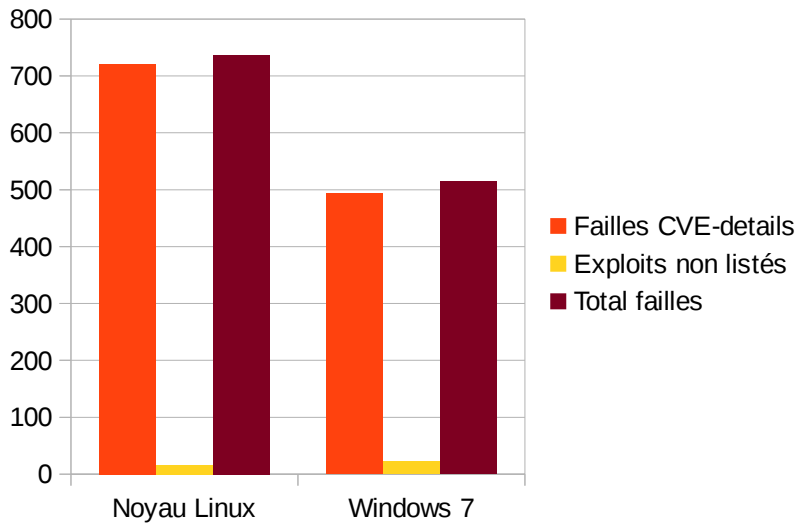


Diagramme de comparaison du nombre total de failles (CVE et exploits non listés) entre le noyau Linux et Windows 7

22. Failles et exploits de Libre Office

22.1. Les failles CVE-Details

2011	2
2012	5
2014	3
2015	5

22.2. Analyse des exploits par rapport aux failles CVE-Details

Date	Listé	Non Listé	Poids		CVE
2012-05-28		*1*		*1*	NON COMMUNIQUE
2012-04-19		*1*		*1*	NON COMMUNIQUE

23. Failles et exploits de Microsoft Office

23.1. Les failles CVE-Details

2010	55
2011	30
2012	19
2013	17
2014	10
2015	40

23.2. Analyse des exploits par rapport aux failles CVE-Details

Date	Listé	Non Listé	Poids		CVE
2015-12-14		*1*	*7*		2015-6132
2015-12-09		*1*	*7*		2015-6128
2015-09-16		*1*	*9*		2015-2523
2015-09-16		*1*	*9*		2015-2520
2015-09-16		*1*	*9*		2015-2521
2015-09-16	*1*		*9*		2015-2510
2015-08-31		*1*		*1*	NON COMMUNIQUE
2018-08-25		*1*	*9*		2015-0065
2015-08-25	*1*		*9*		2015-0064
2015-08-21		*1*		*1*	NON COMMUNIQUE
2015-08-21	*1*		*9*		2015-2469
2015-08-21	*1*		*9*		2015-2431
2015-08-21	*1*				2015-2468
2015-08-21	*1*		*9*		2015-2467
2015-08-21	*1*		*9*		2015-2470
2015-02-28		*1*		*1*	NON COMMUNIQUE
2014-11-12	*1*		*9*		2014-4114
2013-07-01		*1*	*7*		2014-2671
2013-02-20	*1*		*9*		2010-3333
2012-11-20		*1*		*1*	NON COMMUNIQUE
2012-11-09		*1*		*1*	NON COMMUNIQUE
2012-10-29		*1*		*1*	NON COMMUNIQUE
2012-10-28		*1*		*1*	NON COMMUNIQUE
2012-10-25		*1*		*1*	NON COMMUNIQUE
2012-10-24		*1*		*1*	NON COMMUNIQUE
2012-10-11	*1*		*4*		2012-5672
2012-07-31	*1*		*8*		2010-3964
2010-06-11		*1*	*9*		2012-0013
2012-01-08	*1*		*9*		2010-3333
2011-11-22	*1*		*9*		2010-0822
2011-11-05	*1*		*9*		2011-0105
2011-07-03	*1*		*9*		2010-3333
2011-06-26		*1*	*8*		2010-1681

2011-06-14		*1*		*1*	NON COMMUNIQUE
2011-04-29	*1*		*9*		2011-0978
2010-10-16		*1*	*9*		2010-3329
2010-09-11		*1*		*1*	NON COMMUNIQUE
2010-09-11		*1*	*9*		2010-1900
2010-09-08		*1*		*1*	NON COMMUNIQUE
2010-09-08		*1*	*8*		2010-1681
2010-08-25		*1*	*9*		2010-3146
2010-08-25		*1*	*9*		2010-3141
2010-04-06		*1*		*1*	NON COMMUNIQUE

24. Récapitulatif des données de comparaison entre Libre Office et Microsoft Office

	Libre Office	Microsoft Office
Exploits listés	*0*	*16*
Exploits non listés	*2*	*27*
Nombre total d'exploits	*2*	*43*
Exploits sans poids	*2*	*13*
Poids des exploits	*0*	*247*
Poids/100 poids	*0*	*823*
Failles CVE-details	*15*	*171*
Exploits non listés	*2*	*27*
Total failles	*17*	*198*
Nombre de failles	*17*	*198*
Nombre d'exploits	*2*	*43*
Pourcentage	*11.76%*	*21.72%*

Tableau récapitulatif des données entre Libre Office et Microsoft Office

	Libre Office	Microsoft Office
Failles CVE-details	*15*	*171*
Exploits non listés	*2*	*27*
Total failles	*17*	*198*

Tableau récapitulatif des failles CVE et des exploits non listés entre Libre Office et Microsoft Office

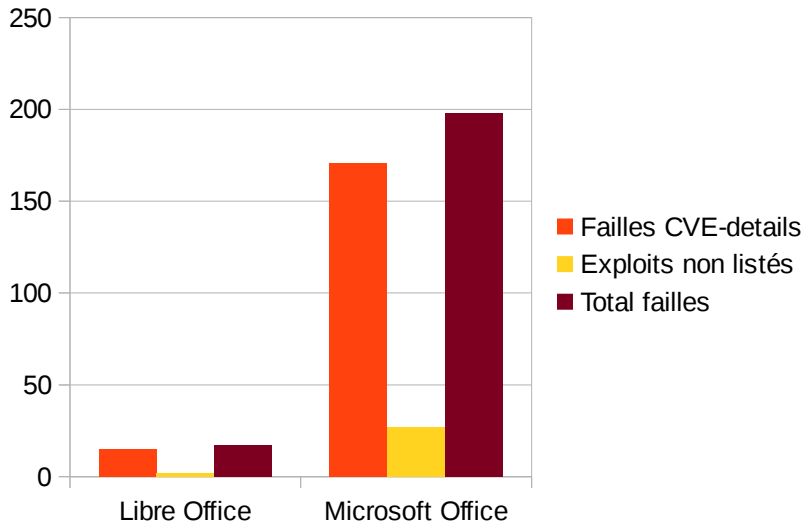


Diagramme de comparaison du nombre total de failles (CVE et exploits non listés) entre Libre Office et Microsoft Office

25. Failles et exploits d'Apache HTTP Serveur

25.1. Les failles CVE-Details

2010	9
2011	12
2012	8
2013	5
2014	11
2015	4

25.2. Analyse des exploits par rapport aux failles CVE-Details

Date	Listé	Non Listé	Poids	Sans poids	CVE
2013-10-04		*1*	*10*		2013-4810
2012-02-06	*1*		*4*		2011-3639
2012-01-31	*1*		*4*		2012-0053
2011-12-09	*1*		*8*		2011-3192
2011-11-24	*1*		*4*		2011-4317
2011-08-19	*1*		*8*		2011-3192

26. Failles et exploits de IIS Serveur

26.1. Les failles CVE-Details

2010	6
2012	1
2013	1
2014	1
2015	1

26.2. Analyse des exploits par rapport aux failles CVE-Details

Date	Listé	Non Listé	Poids	Sans poids	CVE
2015-08-31		*1*		*1*	NON COMMUNIQUE
2014-09-29		*1*		*1*	NON COMMUNIQUE
2012-07-02		*1*		*1*	NON COMMUNIQUE
2012-07-02		*1*		*1*	NON COMMUNIQUE
2012-06-10		*1*		*1*	NON COMMUNIQUE
2012-06-08		*1*	*8*		2002-1142
2011-07-03		*1*	*3*		2009-2521
2011-01-08		*1*	*8*		2001-0333
2010-12-21	*1*		*10*		2010-3972
2010-11-12		*1*	*9*		2009-3023
2010-10-01	*1*		*4*		2010-1899
2010-09-20		*1*	*6*		2005-1734
2010-09-20		*1*		*1*	NON COMMUNIQUE
2010-07-25		*1*	*8*		2003-0349
2010-07-25		*1*	*8*		2003-0822
2010-07-25		*1*	*8*		2003-0109
2010-07-07		*1*	*10*		2004-1134
2010-07-02	*1*		*7*		2010-2731
2010-06-15		*1*		*1*	NON COMMUNIQUE
2010-06-15		*1*	*10*		2001-0500
2010-04-30		*1*	*10*		2000-1089
2010-04-30		*1*	*10*		1999-0874
2010-04-30		*1*	*10*		2001-0241
2010-02-13		*1*		*1*	NON COMMUNIQUE

27. Récapitulatif des données de comparaison entre Apache HTTP Serveur et IIS Serveur

	Apache HTTP	IIS Serveur
Exploits listés	*5*	*3*
Exploits non listés	*1*	*21*
Nombre total d'exploits	*6*	*24*
Exploits sans poids	*0*	*8*
Poids des exploits	*38*	*129*
Poids/100 poids	*633*	*806*
Failles CVE-details	*49*	*14*
Exploits non listés	*1*	*21*
Total failles	*50*	*35*
Nombre de failles	*50*	*35*
Nombre d'exploits	*6*	*24*
Pourcentage	*12.00%*	*68.57%*

Tableau récapitulatif des données entre Apache HTTP Serveur et IIS Serveur

	Apache HTTP	IIS Serveur
Failles CVE-details	*49*	*14*
Exploits non listés	*1*	*21*
Total failles	*50*	*35*

Tableau récapitulatif des failles CVE et des exploits non listés entre Apache HTTP Serveur et IIS Serveur

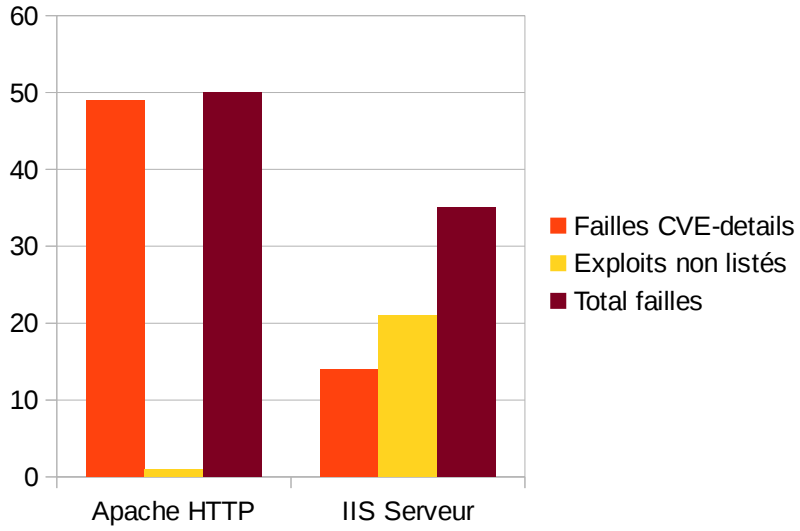


Diagramme de comparaison du nombre total de failles (CVE et exploits non listés) entre Apache HTTP Serveur et IIS Serveur

28. Récapitulatif des données de comparaison entre les logiciels open source et closed source

	NL	HTTP	LO	OS	W7	IIS	MO	CS
Exploits listés	*55*	*5*	*0*	**60**	*20*	*3*	*16*	**39**
Exploits non listés	*16*	*1*	*2*	**19**	*22*	*21*	*27*	**70**
Nombre total d'exploits	*71*	*6*	*2*	**79**	*42*	*24*	*43*	**109**
Exploits sans poids	*14*	*0*	*2*	**16**	*15*	*8*	*13*	**36**
Poids des exploits	*325*	*38*	*0*	**363**	*203*	*129*	*247*	**579**
Poids/100 poids	*570*	*633*	*0*	**576**	*752*	*806*	*823*	**793**
Failles CVE-details	*721*	*49*	*15*	**785**	*493*	*14*	*171*	**678**
Exploits non listés	*16*	*1*	*2*	**19**	*22*	*21*	*27*	**70**
Total failles	*737*	*50*	*17*	**804**	*515*	*35*	*198*	**748**
Nombre de failles	*737*	*50*	*17*	**804**	*515*	*35*	*198*	**748**
Nombre d'exploits	*71*	*6*	*2*	**79**	*42*	*24*	*43*	**109**
Pourcentage	*9.63%*	*12.00%*	*11.76%*	**9.83%**	*8.16%*	*68.57%*	*21.72%*	**14.57%**

Tableau récapitulatif des données entre les logiciels open source et closed source

Légende :

- NL : noyau Linux
- HTTP: Apache HTTP Serveur
- LO : Libre Office
- OS : Open Source
- W7 : Windows 7
- IIS : Serveur IIS
- MO : Microsoft Office
- CS : Closed Source

	Open Source	Closed Source
Failles CVE-details	785	678
Exploits non listés	19	70
Total failles	804	748

Tableau récapitulatif des failles CVE et des exploits non listés entre les logiciels open source et closed source

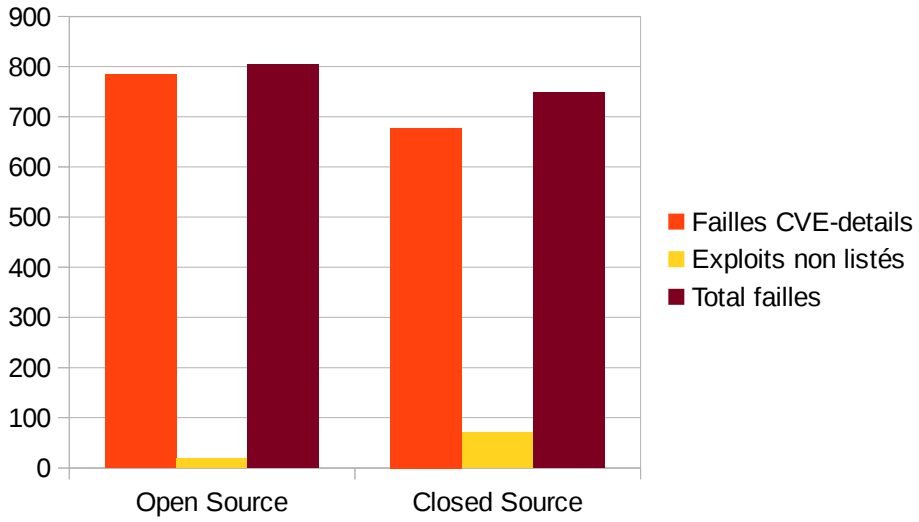


Diagramme de comparaison du nombre total de failles (CVE et exploits non listés) entre les logiciels open source et closed source

29. Analyse des données du noyau Linux

29.1. SE – Phase de développement du logiciel

Non définissable	*0*
Non sécurisé	*0*

Phase numéro	**Choix**
1	*0*
2	*1*
3	*0*

Poids total *1*

29.2. SE – Le pourcentage de patchs réalisés

Non définissable	*0*
Nombre total de failles	*743*
Nombre total de patchs	*1108*
Pourcentage de patchs réalisés	149,13%

29.3. SE – Les exploits réalisés sur un logiciel

Non définissable	*0*
Année Début :	*1997*
Année Fin :	*2016*
Nombre d'années prises en compte	*20*
Nombre d'exploits :	*277*
Moyenne du nombre d'exploits sur un an :	*14*

29.4. SI – Le poids des failles d'un logiciel

Non définissable *0*

Année Début : *2011*
Année Fin : *2015*
Nombre d'années prises en compte *5*

Récapitulatif

Poids	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Nombre de failles	*2*	*98*	*11*	*39*	*244*	*46*	*109*	*37*	*5*	*6*

Nombre total de failles *597*
Moyenne du nombre de failles sur une année *119*
Poids total *3.047*
Moyenne du poids sur une année *609*
Moyenne du poids en % *510*

29.4.1. Détails

Voir le chapitre « *5. Failles du noyau Linux* »

29.5. SI – Les lignes de code du logiciel

Non définissable *0*

Nombre de lignes de code *16.980.546*

Nombre de vulnérabilités découvertes *1397*
Année Début *1999*
Année Fin *2016*
Nombre d'années prises en compte *18*
Moyenne du nombre de vulnérabilités découvertes sur une année *78*

Nombre de vulnérabilités théoriques *485.158*

Pourcentage de vulnérabilités réelles *0,02%*

Source :
https://www.linuxcounter.net/statistics/kernel

Version 4.5.4

30. Analyse des données de Windows 7

30.1. SE – Phase de développement du logiciel

Non définissable	*0*
Non sécurisé	*0*

Phase numéro	**Choix**
1	*0*
2	*0*
3	*1*

Poids total	*3*

30.2. SE – Le pourcentage de patchs réalisés

Non définissable	*0*
Nombre total de failles	*436*
Nombre total de patchs	*92*
Pourcentage de patchs réalisés	21,10%

30.3. SE – Les exploits réalisés sur un logiciel

Non définissable	*0*

Année Début :	*2000*
Année Fin :	*2016*
Nombre d'années prises en compte	*17*

Nombre d'exploits :	*98*

Moyenne du nombre d'exploits sur un an :	*6*

30.4. SI – Le poids des failles d'un logiciel

Non définissable *0*

Année Début : *2011*
Année Fin : *2015*
Nombre d'années prises en compte *5*

Récapitulatif

Poids	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Nombre de failles	*0*	*19*	*2*	*19*	*57*	*6*	*190*	*17*	*104*	*15*

Nombre total de failles *429*
Moyenne du nombre de failles sur une année *86*
Poids total *2.993*
Moyenne du poids sur une année *599*
Moyenne du poids en % *698*

30.4.1. Détails

Voir le chapitre « *6. Failles Windows 7* »

30.5. SI – Les lignes de code du logiciel

Non définissable *0*

Nombre de lignes de code *40.000.000*

Nombre de vulnérabilités découvertes *560*
Année Début *2009*
Année Fin *2016*
Nombre d'années prises en compte *8*
Moyenne du nombre de vulnérabilités découvertes sur une année *70*

Nombre de vulnérabilités théoriques *1142857*

Pourcentage de vulnérabilités réelles *0,01%*

Sources
http://www.informationisbeautiful.net/visualizations/million-lines-of-code/

31. Comparaison noyau Linux et Windows 7

Arguments de sécurité	Logiciel	Poids obtenu	Non définissable	Non sécurisé	Point
Phase de développement du logiciel	*Noyau Linux*	*1*	*0*	*0*	***1***
	Windows 7	*3*	*0*	*0*	***0***
Le pourcentage de patchs réalisés	*Noyau Linux*	*149,13%*	*0*	*0*	***1***
	Windows 7	*21,10%*	*0*	*0*	***0***
Moyenne du nombre d'exploits réalisés sur un an	*Noyau Linux*	*14*	*0*	*0*	***0***
	Windows 7	*6*	*0*	*0*	***1***
Moyenne du nombre de failles sur une année	*Noyau Linux*	*119*	*0*	*0*	***0***
	Windows 7	*86*	*0*	*0*	***1***
Moyenne du poids de failles sur une année	*Noyau Linux*	*609*	*0*	*0*	***0***
	Windows 7	*599*	*0*	*0*	***1***
Moyenne du poids des failles en %	*Noyau Linux*	*510*	*0*	*0*	***1***
	Windows 7	*698*	*0*	*0*	***0***
Pourcentage de la moyenne des vulnérabilités par rapport aux lignes de code	*Noyau Linux*	*0,02%*	*0*	*0*	***0***
	Windows 7	*0,01%*	*0*	*0*	***1***

32. Analyse des données de Libre Office

32.1. SE – Phase de développement du logiciel

Non définissable	*0*
Non sécurisé	*0*

Phase numéro	**Choix**
1	*0*
2	*1*
3	*0*

Poids total *1*

32.2. SE – Le pourcentage de patchs réalisés

Non définissable	*0*
Nombre total de failles	*2*
Nombre total de patchs	*32*
Pourcentage de patchs réalisés	1600,00%

32.3. SE – Les exploits réalisés sur un logiciel

Non définissable	*0*
Année Début :	*2012*
Année Fin :	*2012*
Nombre d'années prises en compte	*1*
Nombre d'exploits :	*2*
Moyenne du nombre d'exploits sur un an :	*2*

32.4. SI – Le poids des failles d'un logiciel

Non définissable *0*

Année Début : *2011*
Année Fin : *2015*
Nombre d'années prises en compte *5*

Récapitulatif

Poids	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Nombre de failles	*0*	*0*	*0*	*4*	*0*	*0*	*5*	*4*	*1*	*1*

Nombre total de failles *15*
Moyenne du nombre de failles sur une année *3*
Poids total *102*
Moyenne du poids sur une année *20*
Moyenne du poids en % *680*

32.4.1. Détails

Voir le chapitre « *7. Failles Libre Office* »

32.5. SI – Les lignes de code du logiciel

Non définissable *0*

Nombre de lignes de code *7.181.255*

Nombre de vulnérabilités découvertes *17*
Année Début *2011*
Année Fin *2016*
Nombre d'années prises en compte *6*
Moyenne du nombre de vulnérabilités découvertes sur une année *3*

Nombre de vulnérabilités théoriques *205.179*

Pourcentage de vulnérabilités réelles *0,00%*

Sources :
https://www.openhub.net/p/libreoffice/analyses/latest/languages_summary

33. Analyse des données de Microsoft Office

33.1. SE – Phase de développement du logiciel

Non définissable	*0*
Non sécurisé	*0*

Phase numéro	**Choix**
1	*0*
2	*1*
3	*0*

Poids total	*1*

33.2. SE – Le pourcentage de patchs réalisés

Non définissable	*0*
Nombre total de failles	*353*
Nombre total de patchs	*10*
Pourcentage de patchs réalisés	2,83%

33.3. SE – Les exploits réalisés sur un logiciel

Non définissable	*0*

Année Début :	*2006*
Année Fin :	*2016*
Nombre d'années prises en compte	*11*

Nombre d'exploits :	*64*

Moyenne du nombre d'exploits sur un an :	*6*

33.4. SI – Le poids des failles d'un logiciel

Non définissable *0*

Année Début : *2011*
Année Fin : *2015*
Nombre d'années prises en compte *5*

Récapitulatif

Poids	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Nombre de failles	*0*	*0*	*0*	*7*	*3*	*0*	*4*	*0*	*102*	*0*

Nombre total de failles *116*
Moyenne du nombre de failles sur une année *23*
Poids total *989*
Moyenne du poids sur une année *198*
Moyenne du poids en % *853*

33.4.1. Détails

Voir le chapitre « *8. Failles Microsoft Office* »

33.5. SI – Les ligne de code du logiciel

Non définissable *0*

Nombre de lignes de code *30.000.000*

Nombre de vulnérabilités découvertes *357*
Année Début *1999*
Année Fin *2016*
Nombre d'années prises en compte *18*
Moyenne du nombre de vulnérabilités découvertes sur une année *20*

Nombre de vulnérabilités théoriques *857.143*

Pourcentage de vulnérabilités réelles *0,00%*

Sources :
https://blogs.msdn.microsoft.com/macmojo/2006/11/02/its-all-in-the-numbers/
Version 2006, pour la version MAC

34. Comparaison Libre Office et Microsoft Office

Arguments de sécurité	Logiciel	Poids obtenu	Non définissable	Non sécurisé	Point
Phase de développement du logiciel	*Libre Office*	*1*	*0*	*0*	***0***
	Microsoft Office	*1*	*0*	*0*	***0***
Le pourcentage de patchs réalisés	*Libre Office*	*1600,00%*	*0*	*0*	***1***
	Microsoft Office	*2,83%*	*0*	*0*	***0***
Moyenne du nombre d'exploits réalisés sur un an	*Libre Office*	*2*	*0*	*0*	***1***
	Microsoft Office	*6*	*0*	*0*	***0***
Moyenne du nombre de failles sur une année	*Libre Office*	*3*	*0*	*0*	***1***
	Microsoft Office	*23*	*0*	*0*	***0***
Moyenne du poids de failles sur une année	*Libre Office*	*20*	*0*	*0*	***1***
	Microsoft Office	*198*	*0*	*0*	***0***
Moyenne du poids des failles en %	*Libre Office*	*680*	*0*	*0*	***1***
	Microsoft Office	*853*	*0*	*0*	***0***
Pourcentage de la moyenne des vulnérabilités par rapport aux lignes de code	*Libre Office*	*0,00%*	*0*	*0*	***1***
	Microsoft Office	*0,00%*	*0*	*0*	***0***

35. Analyse des données d'Apache HTTP Serveur

35.1. SE – Phase de développement du logiciel

Non définissable	*0*
Non sécurisé	*0*

Phase numéro	**Choix**
1	*0*
2	*1*
3	*0*

Poids total *1*

35.2. SE – Le pourcentage de patchs réalisés

Non définissable	*0*
Nombre total de failles	*288*
Nombre total de patchs	*255*
Pourcentage de patchs réalisés	88,54%

35.3. SE – Les exploits réalisés sur un logiciel

Non définissable *0*

Année Début : *1996*
Année Fin : *2013*
Nombre d'années prises en compte *18*

Nombre d'exploits : *21*

Moyenne du nombre d'exploits sur un an : *1*

35.4. SI – Le poids des failles d'un logiciel

Non définissable *0*

Année Début : *2008*
Année Fin : *2015*
Nombre d'années prises en compte *8*

Récapitulatif

Poids	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Nombre de failles	*1*	*0*	*5*	*27*	*23*	*1*	*6*	*5*	*0*	*1*

Nombre total de failles *69*
Moyenne du nombre de failles sur une année *9*
Poids total *337*
Moyenne du poids sur une année *42*
Moyenne du poids en % *488*

35.4.1. Détails

Voir le chapitre « *9. Failles Apache HTTP Serveur* »

35.5. SI – Les lignes de code du logiciel

Non définissable *0*

Nombre de lignes de code *1.805.100*

Nombre de vulnérabilités découvertes *183*
Année Début *1999*
Année Fin *2015*
Nombre d'années prises en compte *17*
Moyenne du nombre de vulnérabilités découvertes sur une année *11*

Nombre de vulnérabilités théoriques *51.574*

Pourcentage de vulnérabilités réelles *0,02%*

Sources :
https://www.openhub.net/p/apache

36. Analyse des données d'IIS Serveur

36.1. Phase de développement du logiciel

Non définissable	*0*
Non sécurisé	*0*

Phase numéro	**Choix**
1	*0*
2	*1*
3	*0*

Poids total	*1*

36.2. SE – Le pourcentage de patchs réalisés

Non définissable	*0*
Nombre total de failles	*25*
Nombre total de patchs	*86*
Pourcentage de patchs réalisés	344,00%

36.3. SE – Les exploits réalisés sur un logiciel

Non définissable	*0*

Année Début :	*1997*
Année Fin :	*2014*
Nombre d'années prises en compte	*18*

Nombre d'exploits :	*119*

Moyenne du nombre d'exploits sur un an :	*7*

36.4. SI – Le poids des failles d'un logiciel

Non définissable *0*

Année Début : *2008*
Année Fin : *2015*
Nombre d'années prises en compte *8*

Récapitulatif

Poids	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***
Nombre de failles	*0*	*1*	*2*	*3*	*1*	*4*	*2*	*2*	*4*	*2*

Nombre total de failles *21*
Moyenne du nombre de failles sur une année *3*
Poids total *135*
Moyenne du poids sur une année *17*
Moyenne du poids en % *643*

36.4.1. Détails

Voir le chapitre « *10. Failles IIS Serveur* »

36.5. SI – Les lignes de code du logiciel

Non définissable *1*

Nombre de lignes de code *0*

Nombre de vulnérabilités découvertes *21*
Année Début *2008*
Année Fin *2015*
Nombre d'années prises en compte *0*
Moyenne du nombre de vulnérabilités découvertes sur une année *0*

Nombre de vulnérabilités théoriques *0*

Pourcentage de vulnérabilités réelles *0,00%*

37. Comparaison Apache HTTP Serveur et IIS Serveur

Arguments de sécurité	Logiciel	Poids obtenu	Non définissable	Non sécurisé	Point
Phase de développement du logiciel	*Apache HTTP Serveur*	*1*	*0*	*0*	***0***
	IIS Serveur	*1*	*0*	*0*	***0***
Le pourcentage de patchs réalisés	*Apache HTTP Serveur*	*88,54%*	*0*	*0*	***0***
	IIS Serveur	*344,00%*	*0*	*0*	***1***
Moyenne du nombre d'exploits réalisés sur un an	*Apache HTTP Serveur*	*1*	*0*	*0*	***1***
	IIS Serveur	*7*	*0*	*0*	***0***
Moyenne du nombre de failles sur une année	*Apache HTTP Serveur*	*9*	*0*	*0*	***0***
	IIS Serveur	*3*	*0*	*0*	***1***
Moyenne du poids de failles sur une année	*Apache HTTP Serveur*	*42*	*0*	*0*	***0***
	IIS Serveur	*17*	*0*	*0*	***1***
Moyenne du poids des failles en %	*Apache HTTP Serveur*	*488*	*0*	*0*	***1***
	IIS Serveur	*643*	*0*	*0*	***0***
Pourcentage de la moyenne des vulnérabilités par rapport aux lignes de code	*Apache HTTP Serveur*	*0,02%*	*0*	*0*	***0***
	IIS Serveur	*0,00%*	*1*	*0*	***0***

Bibliographie

[GNU] Système d'exploitation GNU, http://www.gnu.org

[FSF] Free Software Foundation, https://www.fsf.org/

[Alhazmi & Malaiya & Ray, 2005] Omar Alhazmi & Yashwant Malaiya & Indrajit Ray, Security vulnerabilities in software systems : a quantitative perspective, DBSec '05, http://dl.acm.org/citation.cfm?id=2138953

[Microsoft] Microsoft, https://www.microsoft.com

[Apple OS] Apple Open at the source, http://www.apple.com/opensource/

[OSI] The Open Source Initiative, http://opensource.org

[Furnell, 2010] Steven Furnell, Mac Security : An Apple that can't be bitten ?, Network Security, http://dl.acm.org/citation.cfm?id=2304330

[.NET] The .NET Foundation, http://www.dotnetfoundation.org/

[vulnerabilités libre office] Base de données de Libre Office concernant leur vulnérabilité, https://www.libreoffice.org/about-us/security/advisories/

[vulnérabilités apache http] Base de données d'apache http concernant leur vulnérabilité, https://httpd.apache.org/security/

[CVE details] The ultimate security vulnerability datasource, https://www.cvedetails.com/

[Hoepman & Jacobs, 2013] Jaap-Henk Hoepman & Bart Jacobs, Increased security through open source, Communications of the ACM, http://arxiv.org/abs/0801.3924v1

[Exploit DB] Offensive security exploit database archive, https://www.exploit-db.com/

[Offensive Security] Offensive security, manager of exploit db, https://www.offensive-security.com/